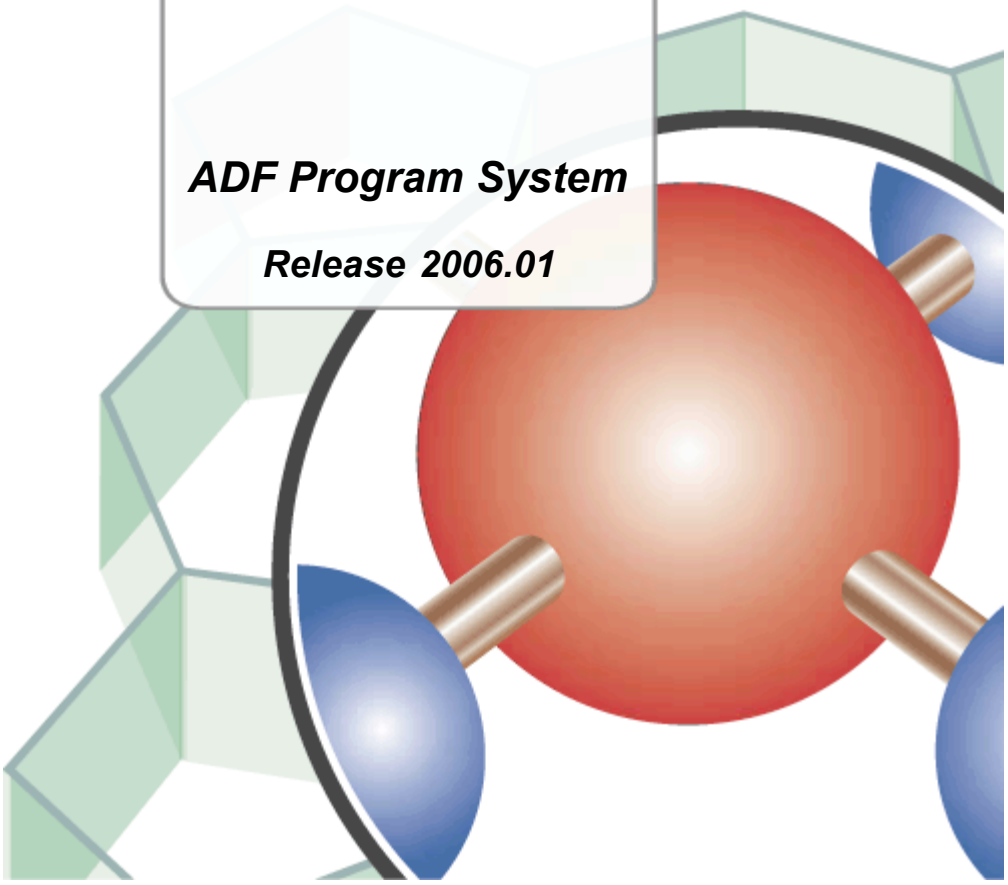
Scientific Computing & Modelling

ADF User's Guide



ADF Program System

Release 2006.01

Scientific Computing & Modelling NV
Vrije Universiteit, Theoretical Chemistry
De Boelelaan 1083; 1081 HV Amsterdam; The Netherlands
E-mail: support@scm.com

Copyright © 1993-2005: SCM / Vrije Universiteit, Theoretical Chemistry, Amsterdam, The Netherlands
All rights reserved

Table of Contents

ADF User's Guide
- **Table of Contents**
- **Preface**
  - ADF 2006.01
- **1 GENERAL**
  - 1.1 Introduction
    - Characterization of ADF
      - Functionality
      - Applicability
      - Model Hamiltonian
      - Analysis
      - Technical
    - Fragments
      - Basic atoms
      - Database
      - Automatic mode
    - Files
      - Standard output
    - Parallel execution
      - File names
  - 1.2 Technical remarks, Terminology
    - Basis functions and orbitals
      - Cartesian function sets, spurious components
      - Frozen core: Core Orbitals and Core Functions
      - Symmetry
      - Orthonormal basis
      - Fragments
      - Summary of functions and orbitals
    - Fit functions
    - Three-step build-up of the bonding
    - Transition State procedure
- **2 INPUT**
  - 2.1 Introduction
    - Running the program
      - The run script 'start'
      - Files produced by ADF
    - Structure of the input
      - Delimiters
      - Uppercase and lowercase
    - Keywords
      - Irrelevant keys, misspelling of keys
    - Minimal input
      - Automatic mode
      - Create mode
      - Fragment mode
  - 2.2 Main Options
    - Parallel Execution
    - Run Types
      - Runtype control and strategy parameters

Atomic Coordinates
    Mixed Cartesian and Z-matrix coordinates
Geometry Optimization
Transition State
Linear Transit
Intrinsic Reaction Coordinate
    IRC start direction
    Forward / Backward IRC paths
Climbing-Image Nudged Elastic Band
Optimization: Special Features
    geovar: constrained optimization, Linear Transit and NEB parameters
    Constrained optimizations: coordinate types
    Constrained optimizations: linear combinations of internal coordinates
    Restrained optimizations
    Symmetry versus constraints
    Z-matrix and symmetry
    Symmetry in a Linear Transit
    Summary of geovar, optim, and atoms
    Initial Hessian
    Hessian values for selected coordinates
Frequencies
    Analytical Frequencies
    Numerical Frequencies
    Accuracy
    Cartesian versus Z-matrix displacements
    Frequencies and GEOVAR keyword
    Isotope Shifts of Vibrational Frequencies
Smoothing of Gradients
Fragments
    Fragment files
QM/MM
Density Functional
    Exchange Correlation Functionals
    Defaults and special cases
        Model potentials
        Hartree-Fock and hybrid potentials
    Meta-GGA and hybrid energy functionals
    Self-Interaction Correction
    General remarks
Relativistic effects
    Pauli
    ZORA
    Spin-Orbit coupling
    Relativistic core potentials
Solvent effects: COSMO
Discrete Solvent Reaction Field Model: DRF
    DRF Theory
    Parameters needed in the DRF model
    DRF input
        EXTERNALS
Frozen-Density Embedding: FDE
    FDE Input
    Preparation of frozen density
    Freeze-and-thaw cycles
    Restrictions and pitfalls

Electric Field: Homogeneous and Point Charges
Orientation of the fields
Symmetry
Bonding energy
Polarizability and hyperpolarizability
MM Dispersion
Time-dependent DFT: Excitation Energies, (Hyper) Polarizabilities
General remarks on the use of the TDDFT Response and Excitation functionality
Excitation Input
Excitation energies for open-shell systems
Spin-flip excitation energies
Core Excitation energies
Excitation energies and Spin-Orbit coupling
Resonance Raman
Resonance Raman for several excited states
Restrictions: (avoided) crossings between excited-states
Restrictions: results not trustworthy for higher excited states
Advanced Restarts
Resonance Raman Input options
Applications of the Excitation feature in ADF
Input description for the Response functionality
Analysis options for TDDFT implementation (excitation energies and polarizabilities)
Time-dependent Current DFT
ESR
EFG
Electronic Configuration
Spin: restricted vs. unrestricted
Unrestricted and Spin-Orbit Coupling
Net Charge and Spin polarization
Orbital occupations: electronic configuration, excited states
CHARGE vs. OCCUPATIONS
Create mode
Frozen core vs. pseudopotentials
Multiplet States
Precision and Self-Consistency
Numerical Integration
Frequencies
Self-adapting precision during optimizations
More integration options
SCF
Interpretation of Input
Units of length and angle
Expressions
Constants and functions
Strings
Where does parsing apply?
Constants vs. geometric parameters
Restarts
Check-point file
General remarks
The restart key
Structure of the restart file
Data on the restart file
SCF data
Coordinates

Hessian
Transition State
Linear Transit
IRC
Frequencies
Printed Output
Print / NoPrint
Debug
Eprint
Eprint subkeys vs. Print switches
Fit
Frag
Freq
GeoStep
NumInt
OrbPop
OrbPopER
Repeat
SCF
SFO
TransitionField
Other Eprint subkeys
Orbital Energies
Mulliken Population Analysis
Population Analysis per MO
Mayer Bond orders
Reduction of output
ASCII Output Files with Atomic Coordinates
2.3 More Options
General
Link-in Input files
Title and Comment
Layout of input
Geometry
Orientation of Local Atomic Coordinates
Symmetry
Ghost Atoms & Non-standard Chemical Elements
Creation
Use as fragment
Basis Set Superposition Error (BSSE)
Hamiltonian
Spin-polarized start-up potential
Unrestricted fragments
Remove Fragment Orbitals
Core Potentials
Properties and Analysis
NMR Chemical Shifts
NMR spin-spin coupling constants
EPR parameters
Localized Molecular Orbitals
Bond order analysis
NBO analysis
Bader's analysis
Precision
Numerical integration

Symmetric density fit
Fit integrals
True density in XC potential
Atomic radial grid
Dependency (basis set, fit set)
Control of Program Flow
Limited execution
Direct SCF: I/O vs. recalculation of data
Skipping
Ignore checks
Parallel Communication Timings
Technical Settings
Memory usage
Vector length
Tails and old gradients
Linearscaling
All Points
Full SCF
Full Fock
Electrostatic interactions from Fit density
Save info
3 Recommendations, problems, Questions
3.1 Recommendations
Precision
Electronic Configuration
Spin-unrestricted versus spin-restricted, Spin states
Geometry Optimization
Bond angles of zero or 180 degrees
Sloppy modes
Step convergence
Basis Sets for Organic Molecules: Single-zeta vs. Double-zeta
Frequencies
Relativistic methods
3.2 Trouble Shooting
License file corrupt
Recover from Crash
Memory Management
Insufficient Space for Allocation
SCF
No convergence
Convergence difficulties with spin-unrestricted calculations
Convergence difficulties with solvation calculations
Geometry Optimization
No convergence
Spurious jumps
Constraints are violated
Clearly wrong results (bond lengths)
Frequencies
Imaginary Frequencies
Geometry-displacement numbers in the logfile are not contiguous
Input ignored
SFO Populations
Error Aborts
Warnings
3.3 Questions

4 FILES
- 4.1 Parallel Execution
- 4.2 Standard output
  - Input Echo, Output Header
  - Main Job Characteristics
  - Build Info: Fragments and Function Sets
  - Technical Parameters
  - Computational Report
  - Results
    - Nuclear and Electronic Configuration
    - ESR Properties
    - Populations, Charge analysis
    - Dipole moment, Quadrupole moment, Electrostatic potential
    - Energy and MO analysis
    - Summary of LT or IRC path(s)
  - Frequencies Results
  - Exit Procedure
  - Logfile
- 4.3 Log file
- 4.4 TAPE21
  - Contents of TAPE21
    - Section General
    - Section Geometry
    - Section Fragments
    - Section AtomTypes
    - Section Properties
    - Section Basis
    - Section Core
    - Section Fit
    - Section Num Int Params
    - Section Symmetry
    - Section Spin_orbit
    - Section Energy
    - Section Point_Charges
    - Section GeoOpt
    - Section TS
    - Section LT
    - Section IRC
    - Section IRC_Forward
    - Section IRC_Backward
    - Section Freq
    - Sections Ftyp n
    - Sections Ftyp n?
    - Section Freq Symmetry
    - Sections X
    - Sections Atyp n X
    - Section LqbasxLqfitx_xyznuc
    - Section GenptData
    - Section Multipole matrix elements
    - Section Irreducible matrix elements
    - Section ETS
  - Using Data from TAPE21
    - Representation of functions and frozen cores
    - Evaluation of the charge density and molecular orbitals
- 4.5 TAPE13

Contents of TAPE13
Section Fit
Section Freq
Section Geometry
Section GeoOpt
Section IRC
Section IRC_Forward
Section IRC_Backward
Section LT
Section TS
5 RESULTS
5.1 Properties
Electronic Configuration, Orbital Energies
Populations and Atomic Charges
Mulliken populations
Atom charges, fragment charges, and bond orders
Bond order analysis
Energy
Thermodynamics
5.2 Plots: Density, Potential, Orbitals
6 APPENDICES
6.1 Database
Data File for Create
Title
Basis functions
Core expansion functions
Core description
Fit functions
Start-up fit coefficients
Example: Calcium
6.2 Elements of the Periodic Table
6.3 Symmetry
Schönfliess symbols and symmetry labels
Molecular orientation requirements
7 References
Keywords
Index

Preface

ADF (Amsterdam Density Functional) is a Fortran program for calculations on atoms and molecules (in gas phase or solution). It can be used for the study of such diverse fields as molecular spectroscopy, organic and inorganic chemistry, crystallography and pharmacochemistry. A separate program in the ADF package (BAND) is available for the study of periodic systems: crystals, surfaces, and polymers.

The underlying theory is the Kohn-Sham approach to the Density-Functional Theory (DFT). This implies a one-electron picture of the many-electron systems but yields in principle the exact electron density (and related properties) and the total energy.

If ADF is a new program for you we recommend that you carefully read Chapter 1, section 1.2 'Technical remarks, Terminology', which presents a discussion of a few ADF-typical aspects and terminology. This will help you to understand and appreciate the output of an ADF calculation.

ADF has been developed since the early 1970s (at that time called HFS, later AMOL), mainly by the two theoretical chemistry groups of, respectively, the Vrije Universiteit in Amsterdam (http://www.chem.vu.nl/en/sec/tc) and the University of Calgary, Canada (http://www.cobalt.chem.ucalgary.ca/group/master.html). Other researchers have also contributed. A recent (new) development center is the Theoretical Chemistry group at the Groningen university (http://theochem.chem.rug.nl). As a major research tool of these academic development groups ADF is in continuous development and retains a firm basis in the academic world.

Maintenance and distribution of the commercial (export) version of the program is done by Scientific Computing & Modelling NV (SCM) (http://www.scm.com), a company based in Amsterdam, formally split off from the theoretical chemistry group in Amsterdam but practically still very much a part of it. Documentation such as User manuals, Installation instructions, Examples, Theoretical documents can be found at the SCM web site.

Publications based on research with ADF should include appropriate references to the program. We recommend that references are made both to the program itself and to publications related to its development and structure. See the 'References' document, available at the SCM web site.

ADF 2006.01

In comparison to ADF 2005.01, the 2006.01 release offers the following new functionality:

- Hartree-Fock and hybrid functionals, such as B3LYP, during the SCF
  (one needs to use the DEPENDENCY key, not available yet in geometry optimizations, NMR, TDDFT, etc.)
- Certain GGA functionals in analytical second derivatives
- Excitation energies including spin-orbit coupling for closed-shell molecules
- New methods for troubleshooting SCF convergence
- ADF recognizes almost 50 solvents by name or formula
- Automatic conversion of the binary KF files

Most important performance improvements:

- Analytical second derivatives are now part of the main ADF program, are faster and easier to use.

Apart from this new functionality and performance improvements, certain bugs have been fixed.

The values of the nuclear electric quadrupole moments and magnetic moments have been updated according to the Handbook of Chemistry and Physics, 86th edition, Editor D.R.Lide, CRC Press, Boca Raton, 2005.

A more extended list of 'what is new or different' can be found in the Updates document.

1 GENERAL

1.1 Introduction

The installation of the Amsterdam Density Functional program package (ADF) on your computer is explained in the Installation manual. This User's Guide describes how to use the program, how input is structured, what files are produced, and so on. Some special applications of ADF are described in the Examples document.

Where references are made to the operating system (OS) and to the file system on your computer the terminology of UNIX type OSs is used and a hierarchical structure of *directories* is assumed.

The ADF package is in continuous development to extend its functionality and applicability, to increase its efficiency and user-friendliness, and of course to correct errors. We appreciate comments and suggestions for improvement of the software and the documentation.

Characterization of ADF

Functionality

- Single Point calculation
- Geometry Optimization
- Transition States
- Frequencies and thermodynamic properties
- Tracing a Reaction Path
- Computation of any electronic configuration
- Excitation energies, oscillator strengths, transition dipole moments, (hyper)polarizabilities, Van der Waals dispersion coefficients, CD spectra, ORD, using Time-Dependent Density Functional Theory (TDDFT)
- ESR (EPR) g-tensors, A-tensors, NQCCs
- NMR chemical shifts and spin-spin coupling constants
- Various other molecular properties
- Treatment of large systems and environment by the QM/MM (Quantum Mechanics / Molecular Mechanics) hybrid approach.

Applicability

All elements of the periodic table can be used (Z = 1-118). For each of the elements, the database contains basis sets of different sizes, ranging from minimal to high quality. Special basis sets are provided for relativistic calculations within the ZORA approach and for response calculations that require additional diffuse basis functions.

Model Hamiltonian

- A choice of Density Functionals, both for the Local Density Approximation (LDA), for the Generalized Gradient Approximation (GGA), for hybrid functionals (currently for potential and energies only), and for meta-GGA functionals (currently for energies only) are available.
- Spin: restricted or unrestricted
- Relativistic effects: scalar approximation and spin-orbit (double-group symmetry),

using the (now recommended) ZORA or the (previously used) Pauli formalism
- Environment: Solvent Effects, Homogeneous Electric Field, Point Charges (Madelung Fields), QM/MM method

Analysis

- Decomposition of the bond energy in 'chemical' components (steric interaction, Pauli repulsion, orbital interactions...)
- Representation of data (Molecular Orbital coefficients, Mulliken Populations) in terms of the constituent chemical fragments in the molecule, along with the conventional representation in elementary basis functions
- Atomic charge determination by Hirshfeld analysis and by Voronoi analysis, multipole derived charges, along with the classical Mulliken populations, and Mayer bond orders

Technical

- The implementation is based upon a highly optimized numerical integration scheme for the evaluation of matrix elements of the Fock operator, property integrals involving the charge density, etc. The code has been vectorized and parallelized.
- Basis functions are Slater-Type Orbitals (STOs). A database is available with several basis sets for each atom in the periodic table of elements.
- The Coulomb potential is evaluated via an accurate fitting of the charge density with so-called fit functions, which are Slater-type exponential functions centered on the atoms. The fit functions are included in the database files.
- A frozen core facility is provided for an efficient treatment of the inner atomic shells.
- Extensive use is made of point group symmetry. Most of the commonly encountered symmetry groups are available.
- Linear scaling techniques are used to speed up calculations on large molecules

Fragments

ADF has a fragment oriented approach: the poly-atomic system to be computed is conceptually built up from fragments, the molecular one-electron orbitals are calculated as linear combinations of fragment orbitals, and final analyses of e.g. the bonding energy are in terms of fragment properties. The fragments may be single atoms or larger moieties.

When you compute a system in terms of its constituent fragments, these fragments must have been computed before and their properties must be passed on to the current calculation. This is done by attaching *fragment files*, which contain the necessary information. A fragment file is simply the standard result file of an adf calculation on that fragment.

When using Basic Atoms as fragments, you do not need to create the fragment files yourself. Instead, you may use the Basis key, and ADF will create the required fragment files automatically. We therefore recommend this feature for starting ADF users.

Basic atoms

Obviously there must be a set of fundamental fragments that are not defined in terms of smaller fragments. Therefore ADF has two modes of execution: the normal mode, using fragments, and the create mode, in which a fundamental fragment is generated. Such a fundamental fragment *must* be a single atom, spherically

symmetric and spin-restricted (i.e. spin-α and spin-β orbitals are spatially identical, they are equally occupied, and fractional occupations are applied, if necessary, to distribute the electrons equally over symmetry-degenerate states). Such a fundamental fragment is denoted a *basic atom*. The basic atoms are the smallest building blocks from which any 'real' calculations are started.

One should realize that the basic atoms are artificial objects that are convenient in the computational approach but that do not necessarily represent real atoms very well (in fact, usually not at all). The bonding energy of a molecule with respect to basic atoms, for instance, should be corrected for this discrepancy in order to get a decent comparison against experimental data. See ref. [1] for a discussion and for examples of applicable values.

A basic atom is computed in the conventional way. The one-electron orbitals are determined as linear combinations of basis functions; the frozen core approximation may be applied for the inner atomic states; a particular type of density functional can be chosen, et cetera. You may have, for instance, different basic Copper atoms by using different basis sets, by choosing different levels of frozen core approximations, or by applying different density functionals.

Database

The ADF package is equipped with a database to help you generate basic atoms. Each data file in the database contains a standard basis set (and related information) for the creation of one basic atom. The data files are relatively small ASCII files. You can easily inspect them. In Appendix 1 a definition is given of such a file. This enables you to create variations and construct your own adapted basis sets. Important: names of the standard basis sets have changed starting from the ADF 2002.01 version to more intuitive names: I→SZ, II→DZ, III→DZP, IV→TZP, and V→TZ2P. Note that in some places the old names are still not replaced by the new names.

The basis functions used in ADF are commonly known as Slater Type Orbitals (STOs). A basis set can roughly be characterized by its size (single-, double-, triple-zeta; with or without polarization) and by the level of frozen core approximation. Initially, the only basis sets provided with ADF were those in the directories I, II, III, IV, V, which now have the more intuitive names SZ, DZ, DZP, TZP, and TZ2P, respectively. The increasing numbers point to an increase in size and quality. It is not possible to give a formally correct short general classification for each basis set directory. However, generally speaking we can say that SZ is a single-zeta basis set, DZ is a double zeta basis set, DZP is a double zeta polarized basis, TZP is a core double zeta, valence triple zeta, polarized basis set, and finally TZ2P is a core double zeta, valence triple zeta, doubly polarized basis. This explains the more intuitive names that are given for the basis sets. The names have also been changed since some of the basis sets have been modified substantially.

In addition, the database contains directories with special basis sets:

ZORA contains basis sets that should be used (exclusively) for relativistic calculations with the ZORA approach. Using 'normal' basis sets in a ZORA calculation may give highly inaccurate results, in particular for heavy elements. The same is classification is used for the directories ZORA/SZ-TZ2P as in the non-relativistic directories. The ZORA basis sets were added later because of the special requirements on basis sets for ZORA relativistic calculations, especially in the core region. The ZORA/QZ4P basis set can be loosely described as core triple zeta, valence quadruple zeta, with four sets of polarization functions.

ET contains several even tempered basis sets which enables one to go to the basis set limit. The accuracy of the smallest basis set in this directory can loosely be described as quadruple zeta in the valence with three polarization functions added. This directory also contains basis sets with extra diffuse functions. In Response calculations one should use such large basis sets. Very diffuse functions are absolutely necessary to get good results for excitation energies corresponding to high lying orbitals.

AUG contains several augmented standard basis sets which enables one to get reasonable results for

excitation energies with relatively small basis sets.

OLD contains basis sets that were contained in the 2.3 release, but that we feel should not be used anymore unless with great care, primarily because the involved frozen cores are too large to justify the frozen core approximation. In some cases we found uncomfortably large errors in equilibrium geometries resulting from such too-large cores.

Furthermore, you will find in the database:

Special/AE contains non-relativistic basis sets for all-electron calculations. However, these files cannot be used as such, because they don't contain any fit sets. Using basis sets without fit sets is pointless and is in fact not possible at all. (The usage and relevance of fit functions is explained later). Therefore, they serve as starting point for the development of (new) basis sets. For some of the all-electron sets appropriate fit sets have already been generated. The corresponding data base files can be found in the appropriate subdirectories SZ, DZ, DZP, et cetera.

Special/Vdiff contains non-relativistic basis sets that include very diffuse functions. These were recommended to be used for Response calculations. Very diffuse functions are absolutely necessary to get good results for excitation energies corresponding to high lying orbitals. Recommendation: use the even tempered basis sets in the ET directory, since these basis sets are better.

Special/MDC contains non-relativistic basis sets with optimized fit functions especially useful for accurate Multipole Derived Charges. These are available only for a limited number of basis sets.

Cerius contains data files that are used in the Cerius2-ADF graphical user interface.

Dirac contains the input files for the DIRAC auxiliary program (see the UTILITIES document)

Band contains input files for the BAND program (see the BAND User's Guide)

ForceFields contains force field files to be used in the QM/MM functionality. Their structure and contents are described in the QM/MM manual. See also the pdb2adf utility (Utilities document), which transforms a PDB file into an ADF input file, for use with QM/MM.

The files in SZ/ (old name I/) (and ZORA/SZ/) have minimal basis sets: single-zeta without polarization. The exponents of the functions correspond to the standard STO-3g basis sets used in programs that employ Gaussian type basis functions. The frozen core approximation is applied, however, for the inner atomic shells. Type-SZ database files are provided only for the lighter elements, up to Kr.

The files in DZ (old name II) can be characterized as double-zeta basis sets without polarization functions. A triple-zeta set is used for the 3d shells of the first row transition metals, the 4f shells of the Lanthanides, and the 5f shells of the Actinides. In all these cases a double-zeta set provides a rather poor expansion basis for the true (numerically computed) atomic orbital.

The basis sets in DZP (old name III) are derived from DZ (old name II), extended with a polarization function. This type of basis sets is thus far provided only for the elements up to Ar, and for the 4p series Ga through Kr.

TZP (old name IV) contains triple-zeta basis sets. A polarization function is added for H through Ar and for Ga through Kr (from DZP).

TZ2P (old name V) finally gives extended basis sets: triple-zeta with two polarization functions, for H through Ar and Ga through Kr (from DZP). Note that the TZ2P database files are provided only for the lighter elements, up to Kr. The ZORA/TZ2P database files are provided for all elements. Typically for all elements one polarization function is added compared to the corresponding TZP basis set. Note, however, that TZ2P will not always give you extra basis functions for most lanthanide and actinide frozen core basis sets.

Multiple occurrences of one chemical element in the same basis set subdirectory correspond to different

levels of the frozen core approximation. Manganese for instance may have a basis set for an atom with a frozen 2p shell and another one with a frozen 3p shell. The file names are self-explanatory: Mn.2p stands for a data file for Manganese with frozen core shells up to the 2p level. An all-electron basis set would correspond to a file that has no frozen-core suffix in its name.

Another type of multiple occurrence of one element in one database directory may be found when basis sets have been developed for different electronic configurations: the Slater-type basis sets are fitted then to numerical orbitals from runs with different occupation numbers. Currently this applies only for Ni (in database directories DZ, TZP and TZ2P), where basis sets are supplied for the d8s2 and the d9s1 configurations respectively. Since in earlier releases only the d8s2 variety was available, the names of the database files are Ni.2p (for d8s2) and Ni_d9.2p, and likewise Ni.3p and Ni_d9.3p.

As mentioned above, some all-electron basis sets are present in the basis set directories SZ through TZ2P (old names I-V), but not for all elements of the periodic table. The heavier elements, from Rb on, the non-relativistic all electron basis sets are missing. In the ZORA basis sets directory you will find all-electron basis sets for all elements (Z = 1-118), which also could be used in non-relativistic calculations. Note, however, that these basis sets were optimized for ZORA calculations, which means that non-relativistic calculations will not always give you the expected accuracy. Warning: the frozen core basis sets in the ZORA directory should never be used in non-relativistic calculations. Non-relativistically optimized basis sets for the heavier elements are provided in a separate directory AE, which contains basis sets of single-, double- and triple-zeta quality indicated respectively by suffixes 'sz', 'dz', and 'tz'. The files in Special/AE/ are not complete database files, because they don't contain fit sets (the usage and relevance of fit functions is explained later).

The development of fit sets and their testing is not a triviality. It is absolutely a bad idea to take a fit set from another database file, corresponding to some frozen core level, and use that in an all-electron basis set: this will give significant errors and make results worthless. In the ZORA directory one can find all-electron basis sets with good fits sets for the heavier elements.

Automatic mode

If you are using 'Basic Atom' fragments only, you do not need to prepare the corresponding fragment files yourself. Instead, add the BASIS block key to the adf input, and ADF will generate all the required fragment files for you. This makes your job scripts and ADF inputs simpler, it ensures that consistent options for the create runs and molecular runs are used, and you will be sure that the fragment files used have been created by the same release of ADF.

Files

Any files produced by the program are generated in the local (working) directory where the calculation runs. If you want to keep them, make sure to move them after the calculation has finished to wherever you want to store them.

Files attached to the job, such as fragment files, are by default also assumed to exist in the local directory. You must take care to move or copy required files to that directory before starting the calculation, or to provide via input adequate information to the program where to find the files. In many cases you can specify a complete path to the file.

Most files that are generated by the program, in particular the standard result file that can be used as a fragment file in other calculations, are *binary* files. A binary file should usually not be moved from one machine to another, i.e. it may not be readable by another machine than the one that generated the file, unless the two machines are of the same type. The ADF package provides utilities to convert the ADF binary result files from binary to ASCII, and vice versa, so that you don't have to regenerate your fragment libraries

when going to another machine. See the utilities document.

Two of the files that are produced by ADF deserve special attention. The first is the general result file TAPE21 (.t21 files). It is a binary file that contains a lot of information about the calculation, such as the one-electron orbitals expressed in the basis functions. It can be used as a fragment file for subsequent calculations (although only TAPE21 files from spin-*restricted* calculations can be used as fragment files). Like all files produced by the program, it is generated in the directory where the job runs. Having done a calculation, you will usually store TAPE21 somewhere under a suitable name so that you can later reuse it, as a fragment file, for a restart, to feed it to an analysis program, and so on.

The second is an ASCII log file, called logfile. It accumulates messages from ADF into a (brief) summary of the run. You can inspect it during the calculation to check how far the calculation has proceeded, whether there are important warnings and so on. At the end of the run this file is copied to the tail of the normal standard output file.

Standard output

ADF is a program that lends itself particularly well for chemical analysis. This is a direct result of the fragment-based approach, where properties of the molecule are related to the properties of the constituent fragments, which is precisely how the chemist thinks. Molecular Orbitals are (optionally) analyzed extensively as how they are composed from occupied and virtual fragment orbitals. This inherently implies a large amount of output. Even computations on small molecules may produce startlingly many pages of output. This is not necessarily so because you can regulate the production of output in detail. Obviously, some kind of *default* production of output had to be implemented. The field of ADF users is so wide and diverse that it is hard to satisfy everybody as regards this default level of output. Depending on your purposes the automatic settings, which determine how much output is generated without instructions to the contrary, may yield boringly many numbers that you just skip through in search for the one value you're interested in, or it may be widely insufficient. Therefore, take notice of the possibilities to regulate output.

Above all, however get familiar with the analysis tools that adf provides to see in what ways these may help to interpret your results. In a later chapter a global description of output is given as it is normally produced. The chapter below gives an introduction in some of the essential features of ADF, which may be sufficiently different from what you are used to in other Quantum Chemistry codes to deserve your attention.

Parallel execution

If a parallel version of ADF has been installed you should be aware of a few special aspects of running ADF in parallel. Partially this depends on the platform and on the installation settings.

First of all, you may specify (by command-line options in the run-script and/or by defining suitable environment variables) explicitly how many parallel processes are to be used. Secondly, you should realize that most of the files that you would have in a single-node run are in a parallel run distributed over the parallel processes. Some parts of the file may be identical across the processes while other parts are not and would only after a recombination yield the data of the corresponding single-node file. The normal result files, (standard output, the logfile and the binary result file TAPE21) are complete at the master process.

File names

A special problem arises when the parallel processes use the same directory to create their files in. Whether or not this is the case depends on the platform. Consult the Installation Manual or ask the person who has installed the program, or someone else who is familiar with parallelization on your machine.

If the parallel processes use the same directory a conflict of names would arise if the different processes all tried to create for instance a file called TAPE21. Therefore, the file names are modified in this situation and all normal file names are appended by _n where n is an integer between 0 and n (n+1 being the total number of parallel processes. The _1 files correspond to kid #1, etc. The program run scripts automatically take care of this aspect by renaming or copying files as required. The program run scripts are part of the ADF package and should be used, as described later in this manual (see also the Examples document), in particular for parallel runs.

Standard output is a special case here: the parent writes its normal ('print') output to standard output while the *kids* each write to a file KidOutput. When file names are modified as discussed above, the kid output files become KidOutput_n. There is also a KidOutput_0 (from the parent) but this file is empty (apart from a dummy record): the parent writes its output to standard output.

1.2 Technical remarks, Terminology

A few words about ADF as regards its technical setup and the names and abbreviations used in this manual. References to these will be made in the discussion of output and print switches.

Basis functions and orbitals

Let us make a clear distinction between (basis) *functions* and *orbitals*, even where these phrases are sometimes mixed up in the traditional terminology. Orbitals are always specific combinations of the basis functions. Orbitals are related to the computed eigenfunctions of some Fock operator or Hamiltonian occurring in the run or in a related preceding calculation. Functions are merely the elementary mathematical entities in which the orbitals are expressed. A Slater Type Orbital (STO), for instance is a function, not an orbital.

The physical meaning of one-electron orbitals in DFT has often been questioned. We believe that they are useful quantities for interpretation, just like the HF orbitals. For a recent discussion see [2].

Cartesian function sets, spurious components

ADF employs Slater-type exponential basis functions centered on the atoms. Such a function consists of an exponential part $\exp(-ar)$ and a polynomial pre-factor $r^{kr}x^{kx}y^{ky}z^{kz}$. A function *set* is characterized by its radial behavior (the exponential part and the power of r, kr) and by its angular momentum quantum number *l*. The functions in such a set consist of all possible combinations $x^{kx}y^{ky}z^{kz}$, such that kx+kx+kz=*l*. These are denoted the *Cartesian* spherical harmonics.

The Cartesian function sets are very suitable for computational manipulations, but they have a drawback. By inspection it is easily verified that a *d*-set consists of 6 Cartesian functions, while there can of course be only 5 true *d*-type functions among them: one (linear combination) of them is in fact an *s*-type function ($x^2+y^2+z^2$). Similarly, there are 10 *f*-type Cartesian functions, 3 of which are in fact *p*-functions. And so on. In ADF all such lower-*l* (combinations of) functions are projected out of the basis and not employed. As a consequence the basis set *size* in the sense of the number of degrees of freedom and hence the number of possible eigenfunctions of the Fock operator is smaller than the number of expansion coefficients that refer to the primitive (Cartesian) basis functions.

The abbreviation BAS is used for references to the elementary Cartesian basis functions.

Frozen core: Core Orbitals and Core Functions

To speed up the computation the innermost atomic shells are kept frozen. The frozen Core Orbitals (CO), which are solutions of a large-basis all-electron calculation on the isolated atom, are expressed in an auxiliary set of (Slater-type) basis functions cor-bas, distinct from the valence set. The core basis set and the expansion coefficients that give the COs expressed in them are stored in the database data files.

Orthogonality of the valence Molecular Orbitals (MO) to the COs is achieved with the help of so-called Core Functions (CF). These functions are included in the valence set but they are not additional degrees of freedom. Each of the normal valence functions is combined with a linear combination of all CFs in the molecule in such a way that the transformed function (cbas) is orthogonal to all frozen COs in the molecule. There are exactly as many CFs as COs so the orthogonality condition for all valence basis functions amounts to the solution of a linear system where the number of conditions equals the number of parameters.

This aspect once more increases the discrepancy between the number of expansion coefficients of an MO

and the number of MOs: the expansion coefficients in the most elementary bas representation run over all bas functions, including the CFs among them. At some places there may, alternatively, be expansions in the core-orthogonalized BAS functions, CBAS, where the CFs do not count anymore: they are included implicitly in the cbas functions.

Symmetry

The Overlap and Fock matrices become block-diagonal by using symmetry-adapted combination of the (C)BAS functions, such that each such function transforms under the symmetry operators as one of the subspecies of the irreducible representations (irrep) of the symmetry group. Symmetry adapted functions are denoted (C)SBAS.

For a given irrep and subspecies not all elementary basis functions can participate in the symmetry adapted combinations. For instance, for an atom in a reflection plane a basis function that is antisymmetric with respect to the reflection cannot be part of any symmetric combination of functions. In particular for higher symmetries the number of BAS functions that are relevant for a subspecies may be considerably smaller than the total number of BAS functions. This is used to cut down expansion lengths, both as used internally in the computation and construction of the Fock matrix, and in printed output. The printed expansion coefficients (in the bas representation) refer only to the participating BAS functions. A defining list of them is printed at an early stage of the run for each of the subspecies.

Orthonormal basis

It is often computationally convenient to use an orthonormal basis. This is constructed from the CSBAS basis by a Lowdin orthogonalization procedure. The resulting symmetry-adapted orthonormal basis is denoted low.

The MOs are computed by diagonalization of the Fock matrix in the LOW representation. The resulting eigenvectors are easily transformed back to any other representation whenever suitable, such as for instance to the primitive cartesian bas representation (including the CFs).

Fragments

Except in Create mode, where a *basic atom* is constructed, the system is built up from fragments and the corresponding fragment files are attached to the run. The program reads from the files the fragment MOs and these are used as (compound) *basis functions* for the molecular calculation. The fragment MOs are called Fragment Orbitals (FO).

FOs belong of course to one of the symmetry representations of the fragment, but not necessarily to a symmetry representation of the new molecule. The FOs are therefore combined into symmetry-adapted combinations, SFOs, to serve as a symmetry-adapted basis in the molecule. These combinations may involve one or more FOs from the same fragment and/or from different fragments. In the latter case the fragments must be symmetry related by one of the operators of the molecule. Symmetry related fragments must of course be identical, apart from their spatial location: they must be of the same fragment *type*.

FOs are naturally orthogonal to the Core Orbitals of their own fragment, but not necessarily to COs of other fragments. By a suitable combination of the SFOs with all CFs in the molecule we obtain the core-orthogonalized symmetry-adapted CSFOs.

The CSFOs can be transformed to an orthonormal basis by a Lowdin transformation. The resulting basis is called low, as above.

Summary of functions and orbitals

In Create mode the (conceptual) approach is:

BAS ⇒ (core-orthogonalization) ⇒ CBAS ⇒ (symmetry) ⇒ CSBAS ⇒ (orthonormality) ⇒ LOW ⇒ (Fock diagonalization) ⇒ MO

In Fragment mode:

FO (=MO from fragment file) ⇒ (symmetry) ⇒ SFO ⇒ (core-orth.) ⇒ CSFO ⇒ (orthonormality) ⇒ LOW ⇒ (Fock diagonalization) ⇒ MO

Acronyms:

`BAS`

Elementary cartesian basis functions, consisting of a radial part (exponential factor and power of r) and an angular part (cartesian spherical harmonic). The complete BAS set contains spurious lower-*l* combinations; these combinations are projected out and not used in the calculation. The BAS set contains also Core Functions.

`SBAS`

Symmetry-adapted combination of BAS functions.

`CF`

Core Function, part of the bas set. The CFs do not represent degrees of freedom in the basis set but serve only to ensure orthogonalization of the valence space to all frozen Core Orbitals.

`CBAS`

Core-orthogonalized elementary basis functions: the true valence (not-CF) BAS functions transformed by adding a suitable combination of the CFs. The total number of CBAS + the total number of of CFs equals the total number of BAS.

`CSBAS`

Symmetry-adapted combination of cbas functions.

`CO`

Frozen Core Orbitals, expressed as linear combinations of an auxiliary corbas basis set. The corbas set plays no role in the further discussion. The corbas functions are *not* the CFs.

The number of COs equals the number of CFs.

`LOW`

Lowdin orthonormalized symmetry-adapted core-orthogonalized basis. In Create mode they are derived directly from the BAS functions, in Fragment mode from the Fragment Orbitals, which are themselves of course expressible in the BAS set.

`FO`

Fragment Orbital: the MO of a fragment calculation, now used as a *basis function* in the molecule of which the fragment is part.

`SFO`

Symmetry adapted combination of FOs.

CSFO

Core-orthogonalized SFO.

Fit functions

Using Slater-type basis functions yields awkward multi-center integrals in the evaluation of the Coulomb potential. This is remedied by employing an auxiliary set of *fit* functions. Like the basis functions, the fit functions are Slater-type exponential functions centered on the atoms. The true density, a sum of *products* of basis functions, is then replaced (approximated) by a linear combination (not products!) of the fit functions. The combination coefficients are called the fit *coefficients*.

$$\rho(r) = \sum_i c_i f_i(r) \quad (1.2.1)$$

The Poisson equation for the fit functions is easily solved, yielding the (approximate) Coulomb potential as an expansion in fit *potential* functions $f_i^c(r)$

$$f_i^c(r) = \int f_i(r') / |r-r'| \, dr \quad (1.2.2)$$
$$V_{Coulomb}(\rho(r)) \approx \sum_i c_i f_i^c(r) \quad (1.2.3)$$

In the SCF procedure the fit coefficients are computed by a least-squares minimization of

$$\int (\rho_{exact}(r) - \rho_{fit}(r))^2 \, dr = \min \quad (1.2.4)$$

with the constraint that ρ_{fit} contain the correct number of electrons. ρ_{exact} is defined as the sum of occupied orbitals (squared and multiplied by the appropriate occupation number). The accuracy of the fit approximation is important and the fit set plays a role similar to the basis set: too few functions (or badly chosen function characteristics) yield inferior results and there is also such a thing as the fit set limit. The fit functions on an atom are consequently an integral part of the definition of the basic atom and they are included in the Create data files. Fortunately, the size of the fit set does not determine the computational effort in such a drastic way as the size of the basis set does. We have chosen therefore to use always fair (though not extreme) fit sets, with the purpose that the effect of fit-incompleteness should in all cases be small enough to be ignored compared with basis set effects, numerical integration errors and Density Functional deficiencies. This does of course depend somewhat on the computed molecule and the studied properties, so a general guarantee cannot be given and, as with basis set effects, one should always have an open eye for possible problems and check the pertaining information in the output file.

One of the most important properties of a molecule is its energy, or its bonding energy with respect to the constituent fragments. The fit incompleteness introduces two types of errors. The first is that, since the Coulomb potential is only approximated, the SCF solution itself, i.e. the set of self-consistent Molecular Orbitals and their energy eigenvalues may be slightly wrong, yielding an error in the charge density and hence in the energy. Since the energy is to first order stable with respect to changes in the mo coefficients this error in the energy can be assumed very small. The second type of error derives from the computation of the energy from the (self-consistent) charge density, via the Coulomb potential. Let

$$\rho \equiv \rho_{exact}(r) = \rho_{fit}(r) + \delta(r) \quad (1.2.5)$$

and

$$V_{fit}(r) = \int \rho_{fit}(r') / |r-r'| \, dr \quad (1.2.6)$$

For the Coulomb energy of the charge density we have

$$2E_{Coul}(r) = \iint \rho(r)\ \rho(r')\ /|r-r'|\ dr dr' = \int \rho(r)\ V_{fit}(r)\ dr + \iint \rho(r)\ \delta(r')\ /|r-r'|\ dr dr' =$$
$$\int V_{fit}(r)\ [\rho(r)+\delta(r)]\ dr + \iint \delta(r)\ \delta(r')\ /|r-r'|\ dr dr' \quad (1.2.7)$$

from which we see that the fit error is corrected to first order (by adding the fit deficiency $\delta(r)$ to the exact charge density when integrating against the fit potential) and that only a second order term remains that cannot be evaluated, the last term in the right-hand-side of (1.2.7).

A fair impression of the fit quality and the importance of the second order error term is obtained by checking

a) the size of the first order correction term $\int V_{fit}(r)\ \delta(r)\ dr$ and *b*) the norm of the deficiency function, $\int \delta^2(r)\ dr$. Both are printed in standard output, at the end of the output of the SCF procedure computational report. They are usually very small, which gives some confidence that the second order fit error can be ignored.

Three-step build-up of the bonding

The approach of ADF is based on fragments. This applies not only in the analysis at the end of the computation but also in the set-up of the program. The computation of the molecule from its constituent fragments takes place in three steps, and these are reflected in the analysis of bond energy components.

First, the (free, unrelaxed) fragments are placed at their positions in the molecule. This implies an *electrostatic* interaction: for each fragment the Coulomb interaction of its undisturbed charge density with the fields of the other fragments.

Next, the Pauli exclusion principle is applied. Even without considering self-consistency the one-electron orbitals of the combined fragments cannot represent a correct one-determinant wave function because the orbitals of different fragments are not orthogonal to one another. The program performs an orthonormalization of the occupied Fragment Orbitals to obtain an antisymmetrized product. This implies a change in the total molecular charge density from the sum-of-fragments to what is called the sum-of-*orthogonalized*-fragments. The corresponding (repulsive) energy term is evaluated separately and is called *Exchange* repulsion, or alternatively *Pauli* repulsion. The phrase *orthogonal(ized) fragments*, if you find it elsewhere in this manual or in the source code of adf, refers to this aspect. The sum of Pauli repulsion and electrostatic interaction is called the *steric interaction*.

The third phase is the relaxation to self-consistency, with of course the ensuing contributions to the bond energy.

Transition State procedure

This phrase stands for an analysis method described in ref. [3] and has no relation to transition states in chemical reactions. An extensive discussion of bond energy analysis by ADF is given in [4, 5]

The energy associated with a change in charge density, say the relaxation to self-consistency from the sum-of-orthogonal-fragments, can be computed by subtracting final and initial energies, each obtained from the corresponding charge density. For purposes of analysis the change in energy de can be reformulated as

$$dE = \int d\mathbf{r}\ \{\ [\rho_{final}(\mathbf{r})-\rho_{initial}(\mathbf{r})] * \int_{\rho_{initial}}^{\rho_{final}} d\rho\ F[\rho(\mathbf{r})]\ \} \quad (1.2.8)$$

$F(\rho)$ is the Fock operator belonging to the charge density ρ

By writing the density difference ρ_{final} - $\rho_{initial}$ a summation over contributions from the different irreducible representations Γ of the molecular symmetry group, an expression is obtained that lends itself for a

decomposition of the bond energy into terms from the different symmetry representations:

$$dE = \sum_{\Gamma} \int d\mathbf{r} \{ \Delta\rho^{\Gamma}(\mathbf{r}) * \int_{\rho_{initial}}^{\rho_{final}} d\rho\, F[\rho(\mathbf{r})] \} \quad (1.2.9)$$

The integral of the Fock operator over the charge density is now approximated by a weighted summation (in fact, a Simpson integration):

$$\int_{\rho_{initial}}^{\rho_{final}} d\rho\, F[\rho] \approx 1/6\, F(\rho_{initial}) + 2/3\, F(\rho_{average}) + 1/6\, F(\rho_{final}) \quad (1.2.10)$$

$$\rho_{average} = 1/2\, \rho_{initial} + 1/2\, \rho_{final} \quad (1.2.11)$$

The term with the Fock operator due to the average charge density has given rise to the phrase *transition state*. To avoid confusion we will often refer to it as to the transition *field.*

The approximate integral (1.2.10) involves two errors. The first, rather obvious, is the approximation of the exact integral in (1.2.9) by the weighted sum in (1.2.10). Except in pathological cases this approximation is highly accurate.

The second error comes from the fact that the Coulomb and XC potentials in the Fock operator are computed from the *fit* density. This is only an approximation to the true density, while in the original bond-energy expression (energy due to the final density minus energy due to the initial density) no potentials occur and the *exact* charge density can be used. As mentioned before, these fit-related errors are usually small. For the XC potential the true density can be used if one includes the keyword EXACTDENSITY.

All such errors in the total bonding energy are easily corrected by comparing the summation over the Γs with the correct value for the total bonding interaction term. The difference is simply added to the total bond energy, so no true error remains. We only have a (correction) term that can't be split in contributions from the distinct symmetry representations. In the printed bond energy analysis such small corrections are 'distributed' over the other terms by scaling the other terms such that their sum is the correct total value.

2 INPUT

2.1 Introduction

Running the program

When ADF has been installed you can run it by supplying appropriate input and starting the 'adf' script, located in $ADFBIN. This script sets up some environment variables, and parses the input to see if anything special needs to be done. For example, if the BASIS key is used the adf script will also execute commands to make the appropriate fragment files. You can use this run script both for the serial and parallel versions of the program. For other programs in the package, there are similar run scripts ('band', 'dirac', and so on).

Running the program using the run script involves the following steps:

- Construct an ASCII input file, say in.
- Run the program by typing (under UNIX):
  `$ADFBIN/adf {-n nproc} <in >out`
  The part between curly brackets is optional, so the shortest application has the format
  `$ADFBIN/adf <in >out`
  Note that the run files in the $ADFHOME/examples directory are UNIX scripts which are executes with:
  `run >out`
- Move / copy relevant result files (in particular TAPE21) to the directory where you want to save them, and give them appropriate names.
- Inspect the standard output file out to verify that all has gone well.

During the run you may inspect the logfile, to see how far the program has proceeded, or whether you should interrupt the calculation.

In the above scheme adf is the name of the run script that invokes the adf.exe program executable. During the installation the script has been put in the same directory where the program executables are generated: $ADFBIN. You may have moved it to another place, or renamed it. We recommend that you adjust your $PATH variable so that you can omit $ADFBIN from the execution command.

To run another program from the ADF suite, just use the appropriate program run script.

The input for the program is read from standard input, and output is written to standard output, redirected in the example above to in and out, respectively.

The part between square brackets is optional and is only meaningful for a parallel program version. The -n flag specifies the number of parallel processes (nproc) to use. If omitted the default applies, which is the value of the environment variable $NSCM, if such variable exist, otherwise it is defined by installation parameters (in the $ADFHOME/settings file, see the Installation Manual).

Irrespective of the -n value, the $NSCM variable value and any installation parameters, two items define an absolute upper bound for the number of parallel processes that can be used. The first is given (PVM) by the so-called virtual parallel machine, i.e. by the number of hosts that constitute the virtual parallel machine. See the PVM manuals. The second item is the maximum number of parallel processes that you allow to run on each particular host that is part of the virtual parallel machine. This information must be stored in a file nodeinfo, located in the $ADFBIN directory. For any host not mentioned in this file, the maximum number of parallel processes on that host is one (1). Note that all hosts to be included must be known to PVM as being part of the virtual parallel machine, otherwise no process can be spawned on that host at all.

Two examples may illustrate this. First: on an IBM SP you should run one process per node. Therefore, no file nodeinfo is needed. Second: on an SGI Power Challenge with 4 CPUs, you should create a file $ADFBIN/nodeinfo, with one record: the name of the host (as it is known to PVM, consult your PVM manual,

or see our Installation manual) followed by one or more spaces and then the number 4.

The program run scripts have, in fact, more flags and arguments, for special usage. You can get a survey by typing

```
$ADFBIN/adf -h
```

The run script 'start'

A note for those who are used to the ADF 2.3 release, where the run script start was used. The start script still exists (in $ADFBIN) and can be used as before (more or less), but some new features (such as the Basis key) will be absent. We recommend that you use the adf script instead of the start script.

Files produced by ADF

The ADF program may generate several output / result files, along with the standard output file. The most important one is TAPE21 (.t21 file), the general result file. TAPE21 contains relevant information about the outcome of the calculation. You may want to store this file somewhere under an appropriate name, for future usage. The meaning of any other files that are produced are explained later in this User's Guide.

We will start now with a discussion of the input file for ADF.

Structure of the input

Much of the general remarks about input for ADF apply also to related property and analysis programs. See the ADF Properties and Analysis documents for details and any differences.

Delimiters

An input record may contain several items. The general rule is that each sequence of characters that does not contain a delimiter is an entity. Delimiters in this context are: 1) the blank or space character ' ', 2) the comma ',' and 3) the equal sign '='.

It is assumed throughout that only characters of the Fortran character set are used.

DO NOT USE TABS IN THE INPUT FILE! The program may *not* see them as delimiters and the effects are hard to predict!

Uppercase and lowercase

Virtually all input items are case-insensitive, but take notice of the obvious exceptions: names of files and directories are case-sensitive.

Keywords

Input for ADF is structured by keywords, in short: keys. A key is a string of characters that does not contain a delimiter (blank, comma or equal sign). Keys are not case sensitive. Input is read until either the end-of-file condition (*eof*) becomes true, or until a record `end input` is encountered, whichever comes first.

(`end input` is not a key.)

Key-controlled input occurs with two formats. In the first you have only one record, which contains both the key and - depending on the case - associated data: the key *argument*:

```
KEY argument
```

The whole part of the line that follows after the key is the argument. It may consist of more than one item.

The alternative format is a sequence of records, collectively denoted as a key *block*. The first record of the block gives the key (which may have an argument). The block is closed by a record containing (only) the word end. The other records in the block constitute the *data block*, and provide information related to the key.

```
KEY {argument}
 data record
 data record
 ...(etc.)...
 ...
end
```

In this manual, when items are optional, such as the argument in the scheme above, they are typed enclosed in curly brackets {}. The { and } characters themselves are not part of the item.

Block type keys may have subkeys in their data block. The subkeys may themselves also be block type keys. The data blocks of block type subkeys, however, do not end with end, but with subend:

```
KEY {argument}
 data
 data
 subkey {argument}
    subkey data
    subkey data
    ...
 subend
 data
 data
 ...
end
```

Layout features such as an open line, indentation, or the number of spaces between items are not significant.

The format to be used for a key is not optional: each admissible key corresponds to one specific format. As a general rule, the block keys control lists of data, such as atomic position coordinates.

A few special keys can have either format. For such keys the format actually in effect depends on the presence of the argument: the block type applies in absence of the argument. The block type applies also when an argument is present that ends with a continuation symbol. The continuation symbol is the ampersand (&) or, alternatively, two contiguous plus-characters preceded by at least one blank (++):

```
KEY {argument} &
 data
 data
end
```

The various types of keys are referred to respectively as *simple* keys, *block* keys, and *general* keys.

A considerable number of keys can be used to specify the geometry, the model Hamiltonian, cf. the Density Functional, the precision of the calculation, and so on. The order in which keys occur in the input file is immaterial, except that a few special keys determine how input data is interpreted, such as the unit-of-length for atomic coordinates. These *interpretation keys* must be used before the pertaining data in input occur. This will be mentioned explicitly again where they are discussed.

The items that can be addressed with keys and the keys themselves are listed in the Index.

Irrelevant keys, misspelling of keys

Specification of a key that is not relevant in the calculation will go unnoticed. Similarly, if you misspell a key such that it is not recognized, the incorrectly labeled input data will be ignored and the program will proceed as if the intended key had not occurred. This results in the application of pre-defined default values or in an error abort, depending on the case. Therefore, whenever the output suggest that part of your input has been ignored, check the spelling.

In this context we stress again: be alert on TAB characters: don't use them at all.

ADF may recognize a key if it is spelled incompletely, that is, if only some initial substring is given, and also if redundant characters are typed after the end of the key. The reason is that often only a small initial part of the true keyname is checked against the input items. Don't rely on this, however: it is not formally supported and it may get disabled in a next release without further notice.

We advise therefore to stick to the correct key names. In particular, you must avoid to use different abbreviated or elongated forms when a key occurs more than once in input: adf will likely assume that you want to indicate distinct keys and it will associate only one of them with the key you had in mind.

Minimal input

Most keys are optional. Default values apply for omitted keys. Assuming that the defaults are sensible, short input files can often be used. We will examine first the minimal input that is required to run ADF. Having read that part, you can start to do calculations.

Automatic mode

The following input will run a geometry optimization on water, using a (almost) minimal input:

```
ATOMS
 O  0  0  0
 H  1  1  0
 H -1  1  0
End

Basis
End

Geometry
End
```

This is the input for the ADF program. You need to store it in a file, and pass it as standard input to ADF.

For example, assume you have stored the above input in a file in. Also assume that the $ADFBIN directory is in your $PATH. Then you run ADF using the following command:

```
adf <in >out
```

ADF will run, and the resulting output will be stored in the file out. If you examine the contents of this file, you will find that ADF has actually run three times: two create runs, and one geometry optimization. The fragment files produced by the create runs are saved in t21.H and t21.O, for hydrogen and oxygen respectively.

The ATOMS block key specifies the starting geometry.

The GEOMETRY key instructs ADF to perform a geometry optimization.

The BASIS block key instructs ADF to run the appropriate create runs automatically, using default values for the basis sets to use. For the XC potentials invoked with the MODEL subkey (cf. XC input block) the XC potential in the Create run will be a GGA potential rather than such a model potential, as these potentials cannot currently be applied in Create runs.

The Automatic mode will be used when the Basis key is present in the input:

```
BASIS
 Type bastyp
 Core coretyp
 Path apath
 Atom atompath
 ...
End
```

All subkeys are optional. For most calculations you need only to set the Type and Core subkeys.

`Type bastyp`

`bastyp` is the type of basis set to use, and must correspond with the name of the directory as used within $ADFRESOURCES, or within $ADFRESOURCES/ZORA for ZORA calculations.
Thus valid types are for example: SZ, DZ, DZP, TZP, TZ2P, ...
If no basis set of the specified type is available, ADF will try to use a larger basis set.
Default: DZ.

`Core coretyp`

`coretyp` is the type of frozen core to use. Allowed values: None, Small, Medium, Large.
If no basis set with core is available, an all electron basis set will be used.
If there is only one basis set with core, Small, Medium and Large are identical.
If there are two basis sets with core, Medium and Large are identical.
Default: Large.

`Path apath`

`apath` is an alternative directory with basis sets to use. ADF looks for appropriate basis sets only within this directory.
Default: $ADFRESOURCES

`Atom atompath`

In this subkey 'Atom' should be replaced by the name of the atomic fragment for which you want to specify the basis, for example 'O'.

Use this key to specifically select a basis set for this atom:

- an absolute path to a basis file (for example $ADFRESOURCES/DZ/O.1s)

- a relative path to a basis file (for example DZ/O.1s)
- a filename within the Type directory (for example O.1s)

An absolute path will always be used as specified.
A relative path is relative to the value of the PATH (or PATH/ZORA) subkey.
A filename is always relative to PATH/Type or PATH/ZORA/TYPE directory.

The relative path or filename will automatically switch to a ZORA basis set in case of ZORA calculations.
You can have one Atom subkey for each basic atom type in your input.

Since you pick explicitly the file to use, you are responsible for choosing a reasonable basis set.

Do not include the Fragments or Corepotentials keys when using the Basis key!

When the `Basis` key is present, ADF will first create fragment files for all the basic atom fragments found in the ATOMS key block. Normally this means that for each atom type in your molecule a fragment file will be created.

You may have different fragments with the same atom: add a dot and a name (without spaces) after the name of the element, as described in the ATOMS key. For example: H.1 and H.2. In this example two fragment files will be created: one for the H.1 fragment and one for the H.2 fragment. Using the ATOM subkey you may assign different basis sets to these fragments. Another consequence is that the H.1 and H.2 atoms will never be symmetry equivalent to each other.

Starting from ADF2006.01 the BASIS key recognizes elements denoted with Gh.atom in the ATOMS key as being ghost atoms. If one does not specifically select a basis set for this ghost atom, the all electron basis set for the atom is selected in the creation of the ghost atom using the type of basis set chosen with the BASIS key. The atom name must begin with the standard one- or two-character symbol for the chemical element: Gh.H, Gh.He, Gh.Li, and so on. Optionally it may be appended by .text, where text is any string (not containing delimiters). Examples: Gh.H, Gh.Mn.3, Gh.Cu.dz-new.

The basis set to use follows from the subkeys (or the default values), the XC potential follows from the XC block if it is present in the input.

In case of a relativistic calculation, the DIRAC program will also be run automatically, and the create runs will include the correct relativistic key and corresponding basis sets. For ZORA calculations, ADF first tries to locate a special ZORA basis set. If this does not succeed it will use a normal basis set if the required basis set does not use a frozen core.

The resulting fragment files will be named t21.atom, with 'atom' replaced by the names of the basic atoms present. In case of a relativistic calculation, the corepotentials will be stored on t12.rel.

Create mode

In Create mode the input file can be extremely simple. First, the geometry is trivial: one atom at the origin. Indeed, no coordinates etc. are read from input; any such items are ignored.

Second, the problem is computationally so simple that default settings for precision aspects, such as convergence criteria and levels of numerical integration accuracy, are internally defined to be much more stringent than in normal calculations. These aspects don't have to be looked after.

In Create mode you need only a one-line input file of the following form:

```
CREATE Atomtype Datafile
```

`Create`

is the keyword. The remainder of the record (atomtype datafile) is the argument.

`Atomtype`

is a name for the basic atom that you want to create. The program reads and interprets this name. Therefore, the name must begin with the standard chemical symbol (H, He, Li, ...) of the element to be created. Optionally the name may have an suffix of the form .text. The suffix begins with a period (.); the part after the period (text) is at your discretion as long as it does not contain a delimiter. A few examples:

appropriate names	*inappropriate names for an atom type*	
K	Si-with-core	: no period after the chemical symbol
Li.newbasis	$HOME/atomicdata/C.dzp	: not beginning with the chemical symbol
P.1992/Feb./30	Ga.nocore,smallbasis	: contains a comma (a delimiter)
	Sodium.2s	: Sodium is not the *symbol* for this element (Na)

Examples of appropriate (left) and inappropriate (right) atom type names used with the keyword `create`.

`Datafile`

specifies the data file that contains the basis set and related items. It may contain a full path if the file does not reside in the working directory of the job.
The `datafile` part is optional. If you omit it, ADF assumes that the file name is identical to the atom type name, i.e.
`Create Atomtype`
is equivalent to and interpreted as
`Create Atomtype Atomtype`
In view of the restrictions that apply to the atom type name, the option to use the short form can only be used if the file name has the appropriate format.
To make the input file easier to understand for a human reader you may, for Datafile, also type file=Datafile, where file= must be typed as such, and datafile is the name of the file.

So you could have a very simple calculation as follows (the 'creation' of a Carbon atom);

```
$ADFBIN/adf << eor
 Create C.dzp
eor
```

The presence of the keyword `create` sets the computational mode of ADF to: *create a basic atom*. The argument (C.dzp) is then analyzed and found to have as initial part C, telling ADF that we'll be creating a Carbon atom. Since the file-specification part is missing, the data file with the basis set etc. must be the (local) file with the name C.dzp.

More often you will directly address a file (with the basis set) that is not local, but located in the database of your ADF package. The script could then be:

```
$ADFBIN/adf << eor
 Create C $ADFHOME/atomicdata/DZ/C.1s
eor
```

Here you address the file 'C.1s' in the database subdirectory DZ/ (this contains basis sets of double-zeta quality).

A considerable number of data files are included in the ADF database. To apply such a file for the creation of a basic atom:

Make a copy of the data file in the directory where you want to run the program. Since the standard data file names satisfy the requirements for atom type names you can now use the simplest option to use the `create` key:
Construct a one line input file *in* (`create` name-of-data-file-copy)

Run ADF by typing
`adf <in >out`
When the calculation has finished, give the result file TAPE21 a suitable name and move it to a directory where you build your database of fragment libraries.
Examine logfile and out to check that everything has gone well.

You may want to define alternative basic atoms, different from those in the standard ADF database, for instance to try out a different basis set developed by yourself. By inspection of one of the standard data files you can see what the contents of such a file should be. A complete description is given in Appendix 1.

You can also create basic atoms corresponding to so-called *Alternative Elements*, with for instance a non-integer nuclear charge or a different mass. See the section Geometry in Chapter 2.3.

Fragment mode

In Fragment mode more input is required than in Create mode: you have to specify at least: (1) the atomic positions and (2) how the total system is built up from fragments. We recommended to specify also (3) the point group symmetry.

Example of an input file for the C2H4 molecule:

```
ATOMS
 C  0    0  .6685
 C  0    0 -.6685
 H  .927 0 -1.203
 H -.927 0 -1.203
 H  .927 0  1.203
 H -.927 0  1.203
end

fragments
 C TAPE21c.dzp
 H TAPE21h.dzp
end

symmetry D(2h)
end input
```

Three keys are used: atoms, fragments and symmetry. The first two are block keys.

`atoms`

defines the atomic positions: each record in the data block contains the chemical symbol of an atom followed by its Cartesian coordinates in Angstroms.
Z-matrix type input of atomic positions is also possible. This will be explained in a later section.

`fragments`

lists the fragment files each record contains a fragment *type* followed by the corresponding fragment *file*. In the example the files are *local* files. Files in other directories are addressed by giving the complete file path. Note: if a *parallel* calculation is performed, be sure that each 'kid' finds the specified fragment files. This will usually require that the files are *not* local to the job, but first be moved to some shared volume, and that the references to the fragment files in the input contain the full path. An alternative is to ensure that the (local) files in the parent directory are copied first to the 'kid' directories before the parallel calculation starts.

`symmetry`

specifies the point group symmetry by a Schönfliess type symbol. Appendix 3 contains a complete list of all Schönfliess symbols that are recognized by adf. If no symmetry is specified adf will take the true symmetry of the nuclear frame as the *user-specified* symmetry. If (electric) fields are used, see later, the computed symmetry will take this into account. Note that the computed symmetry may not occur in the list of allowed symmetries (see Appendix 3), in which case you have to explicitly specify the (lower) point group symmetry you wish to apply.

The atomic coordinates must conform to the point group symmetry; the program will check this and abort if the atomic system does not have the specified symmetry. It is allowed, however, to specify a *lower* symmetry than what is actually present in the set of atomic positions. The *specified* symmetry determines how results are analyzed and how irreducible representations and subspecies are labeled. It also determines various algorithmic aspects: the program runs more efficiently with the highest possible symmetry.

The spatial orientation of the molecular coordinate system is not arbitrary. ADF requires for each pointgroup symmetry a specific standard orientation. In axial groups for instance, the main rotation axis must be the z-axis. This implies a restriction on how you can define the atomic coordinates under atoms. The orientation requirements for all point groups are listed in Appendix 3. If the specified symmetry equals the true symmetry of the nuclear frame adf will adjust the input orientation of the molecule to the requirements (if necessary). If you have specified a subgroup of the true nuclear symmetry, no such orientation adjustment is carried out and the user has to make sure that his input data yield the correct orientation, lest an error will occur.

Restrictions apply to the symmetry (as specified) of the molecule, related to the symmetries of the fragments as they were stipulated in the preceding fragment calculations. All symmetry operators of the molecule that internally rotate or reflect a fragment but leave it at the same position in the molecule, must also be operators of the symmetry group in which the fragment has been computed. Furthermore, two fragments must not be symmetry-equivalent in the molecule only by an improper rotation. The implied internal reflection of the fragment must be one of the symmetry operators in the point group symmetry that is used in the fragment calculation *and* the molecular symmetry group must also contain a proper rotation that maps the two fragments onto each other.

The example of the C2H4 molecule implicitly assumes that all fragments are *single atom* fragments. When the fragments are larger the data records in the atoms key have to be extended: you must specify which atoms belong together in one fragment.

```
SYMMETRY T(D)
Atoms
 Ni 0     0     0
 C -1.06 -1.06  1.06 f=CO/1
 C -1.06  1.06 -1.06 f=CO/2
 C -1.06  1.06 -1.06 f=CO/3
 C  1.06 -1.06 -1.06 f=CO/4
 O  1.71  1.71  1.71 f=CO/1
 O -1.71 -1.71  1.71 f=CO/2
 O -1.71  1.71 -1.71 f=CO/3
 O  1.71 -1.71 -1.71 f=CO/4
End
Fragments
 CO TAPE21co.yesterday
 Ni t21ni.dzp
End
End Input
```

Another sample input file; using a single atom Ni fragment and four molecular CO fragments. The keys symmetry and fragments operate as before. Again we have two types of fragments (here: Ni and CO); for each of them, the fragment file is specified.

Under the key ATOMS the chemical symbols and the nuclear coordinates are listed. Added is the f=...-part; f stands here for fragment and tells the program that the carbon and oxygen atoms belong to CO fragments. The last part /n enumerates the individual CO fragments: here you define which C and O belong together in one CO fragment.

The record for Ni contains no f= part, implying the *default* for this atom: it is a fragment on its own. In the C2H4 example before the default applied to all atoms.

Note that one should use the f= part for symmetry equivalent fragments. In the next example, ADF assumes the fragments CO1, CO2, CO3, and CO4, to be of different fragment types, even though they are coming from the same TAPE21. Therefore ADF will assume symmetry NOSYM in the next calculation, and will not run in T(D) symmetry.

```
Atoms
 Ni 0     0     0
 C -1.06 -1.06  1.06 f=CO1
 C -1.06  1.06 -1.06 f=CO2
 C -1.06  1.06 -1.06 f=CO3
 C  1.06 -1.06 -1.06 f=CO4
 O  1.71  1.71  1.71 f=CO1
 O -1.71 -1.71  1.71 f=CO2
 O -1.71  1.71 -1.71 f=CO3
 O  1.71 -1.71 -1.71 f=CO4
End
Fragments
 CO1 TAPE21co.yesterday
 CO2 TAPE21co.yesterday
 CO3 TAPE21co.yesterday
 CO4 TAPE21co.yesterday
 Ni t21ni.dzp
End
End Input
```

There are more possibilities with the keys atoms and fragments. This is worked out later. The purpose of this section was to provide a quick and easy start.

2.2 Main Options

In the sections and chapters below all keys are discussed in detail. The keywords are typed in small capitals, subkeys in *italic small capitals.* Schematic examples illustrate how the keys are used, and which keys are block keys or general keys.

Structures like 'key=value' should be read as: type 'key=' as such, followed by a suitable value.

Different allowed / eligible values are separated by a bar (|).

Brackets {} around an item, argument, or value indicate that it is optional.

We proceed with a discussion of the most important keys: keys to set the precision of the calculation, keys to regulate the model Hamiltonian (in particular the Density Functional) and keys to specify and control the *run type*.

Parallel Execution

Control of parallelization on the first line of the input file, as in release 2.3, has been disabled in ADF1999 (it is ignored). You now control the parallelization by (a) environment variables and (b) command line argument to the program run script. See the discussion in the introduction, the Installation manual and the Examples document.

Run Types

The different *run types* are characterized by how the geometry is manipulated:

`SinglePoint`

The SCF solution is computed for the input geometry.

`GeometryOptimization`

The atomic coordinates are varied in an attempt to find a (local) energy minimum. One may let all coordinates free or only a subset, keeping the others frozen at their initial values.

`TransitionState`

Search for a saddle point. Similar to a GeometryOptimization, but now the Hessian at the stationary point presumably has one negative eigenvalue.

`LinearTransit`

The geometry is modified step by step from an initial to a final configuration. All of the coordinates or only a subset of them may be involved in the transit. The coordinates to be modified are the LinearTransit *parameters*. For each of the LinearTransit points (geometries) the computation may be a Single Point SCF calculation or a GeometryOptimization. In the latter case only those coordinates (or a subset of them) can be optimized that are not LinearTransit parameters. The LinearTransit feature can be used for instance to sketch an approximate reaction path in order to obtain a reasonable guess for a transition state, from where a true TransitionState search can be started.

`IRC or IntrinsicReactionCoordinate`

Tracing a reaction path from a transition state to reactants and/or products. A fair approximation of the transition state must be input. The end-point(s) - reactants / products - are determined automatically.

`Frequencies`

Computation of force constants and from these the normal vibrational modes and harmonic frequencies. The force constants can be calculated by numerical differentiation of the energy gradients at the equilibrium geometry and the slightly deviating geometries (making small displacements of the atoms). There is however also a possibility with the post-ADF program SD, to calculate the second derivatives analytically. This should usually be faster and in some cases more robust (no SCF convergence problems in displaced geometries, as only a single SCF is required). Note that this program still has some limitations. Most importantly, it can only handle X$alpha and VWN (LDA), but not GGA, calculations at this moment.

`CINEB`

Calculation of the reaction path and transition state search using Climbing-Image Nudged Elastic Band method. This method is further referrer to as NEB or CI-NEB. Using this method one can find a transition state between two known states, further referred to as initial and final states. The choice which state is initial and which is final is arbitrary. During calculation with this method, a number of replicas, or images, of the system is calculated. These images can be considered as being linked by an elastic band. Each image is optimized in such a way that on each step the forces parallel to the reaction path are removed and spring forces are added that keep distances to this image's neighbors equal. At the end of the optimization the images are evenly distributed along the reaction path, the image highest in energy being the transition state (if the climbing-image option is on, the default).

For all features that involve changes in geometry, i.e. all run types except the SinglePoint, it is imperative that you use single-atom fragments. Larger molecular fragments can only be applied in SinglePoint calculations.

Three keys are involved in the specification of the geometry and its manipulation:

`atoms`

sets the atomic (starting) positions.

`geometry`

Controls the run type and strategy parameters, such as convergence thresholds and the maximum number of geometry steps to carry out.

`atoms and geometry`

These two keys together are sufficient for a straightforward Optimization, TransitionState search, IRC run or a Frequencies computation. (Of course, you also need to specify the Fragments or BASIS key.

`geovar`

May be used to impose constraints, for instance when only a subset of all coordinates should be optimized. GeoVar may also be used in a LinearTransit or NEB run to define the LinearTransit or NEB parameters, respectively, and their initial and final values.

Constraints and LinearTransit parameters may also be controlled within the atoms block if a MOPAC-style input format is used, see below.

Runtype control and strategy parameters

With the block key GEOMETRY you define the runtype and strategy parameters.

```
GEOMETRY {RunType { RunTypeData}}
 RunType {RunTypeData}
```

```
End
```

`RunType`

Can be:
- SinglePoint or SP
- GeometryOptimization or GeoOpt or GO
- TransitionState or TS
- IntrinsicReactionCoordinate or IRC
- LinearTransit or LT
- Frequencies or FREQ
- CINEB

If omitted the run type is GeometryOptimization.
If the key GEOMETRY is not used at all the run type is SinglePoint.
The run type specification can be given as argument to the geometry key, or in the data block, but not both.
For some run types additional data may be given after the run type specification.

`RunTypeData`

(Optional) further specifications, depending on the run type. See the sections below.

Omission of the GEOMETRY key altogether effectuates a SinglePoint calculation. A straightforward optimization, with all features that can be set with geometry at their default values, is activated by supplying the key with an empty block:

```
GEOMETRY
End
```

More subkeys are available in the geometry block than just the run type specification. They are used to control strategy parameters such as convergence criteria. All subkeys are optional: default values take effect for those omitted. Some of the subkeys are only meaningful for certain run types. They will be ignored for other run types.

The initial approximation of the Hessian matrix, which may affect the number of optimization steps that are carried out to reach convergence, is not controlled by the key GEOMETRY, but by the key HESSDIAG, and/or by the key GEOVAR. See the section Initial Hessian.

The convergence in geometry optimizations and frequency runs can be improved by smoothing the generation of integration points (implemented in ADF 2004.01), see the section Smoothing of Gradients.

Atomic Coordinates

The input of (initial) atomic positions as Cartesian coordinates has been mentioned already in the minimal-input examples in Chapter 2.1. Alternatively they may be given in z-matrix form.

```
ATOMS {Cartesian / Zmatrix / MOPAC}
 {N} Atom Coords {F=Fragment}
 ...
End
```

`Cartesian or Zmatrix or MOPAC`

Specifies the type of coordinates. Default (no specification) is Cartesian. Instead of Zmatrix you may also type internal.
MOPAC is a special variety: the subsequent records in the data block are MOPAC style Z-matrix input for the atomic system, see example below.

`N`

This is an optional integer by which you may number the atoms. The numbers should be 1,2,3, et cetera if any reference is made to them in other parts of input. The reason for this restriction is that ADF numbers the atoms internally according to their occurrence in the input file and it applies this internal numbering when any subsequent references are interpreted.

`Atom`

The name of an *atom type*. It must begin with the standard one- or two-character symbol for the chemical element: H, He, Li, and so on. Optionally it may be appended by .text, where text is any string (not containing delimiters). Examples: H, Mn.3, Cu.dz-new.

Dummy atoms may be useful in the construction of a Z-matrix, for instance to obtain a set of internal coordinates that reflect the symmetry of the molecule better. They may also be useful in a Z-matrix to avoid an ill-defined dihedral angle, which occurs when three (almost) co-linear atoms span either of the two planes that define the angle. In geometry optimizations this must absolutely be avoided if such internal coordinates are used as optimization parameters.
Dummy atoms are input with the chemical symbol xx. XX-type atoms can be inserted in the list of atoms like any other atom types. The name (xx) can have a suffix of the form *.text.* No fragment files must be supplied for dummies. There are no symmetry constraints on the positions of the dummies. The dummies serve only to set up the Z-matrix in a proper way.

`Coords`

This specifies he coordinates of the atom. If Cartesian coordinates are used the x, y, z values must be given. For Z-matrix coordinates you put first the three connection numbers, then the values of the bond length, bond angle and dihedral angle. Example:

Ge 2 1 5 2.1 95.3 24.8

defines that a Germanium atom is located with a distance 2.1 Angstrom from the second atom in the input list, that the angle (Ge-atom2-atom1) is 95.3 degrees and that the dihedral angle between the planes (Ge-atom2-atom1) and (atom2-atom1-atom5) is 24.8 degrees.

To avoid any confusion as regards the direction (sign) of the dihedral angle, here is the definition used in ADF: Let the connection numbers for an atom P refer to the atoms Q, R and S, in that order. Choose a local coordinate frame such that Q is at the origin, R on the positive z-axis and S in the xz-plane with a positive x-value. The three Z-matrix coordinates bond length, bond angle and dihedral angle of P are then precisely its spherical coordinates r, q, and -f: the distance to the origin, the angle that PQ makes with the positive z-axis (0.**TT**) and the *negative* of the angle that the projection of PQ on the xy-plane makes with the positive x-axis (0..2**TT**, or -**TT**..+**TT**).

The connection numbers and internal coordinate values of the first atom in a Z-matrix have no meaning. Similarly, the second atom requires only a bond-length specification and the third atom only a bond length and a bond angle. However, for each atom three connection numbers are read from input and interpreted, and you must therefore supply *zeros* for them if they don't refer to any atoms. The corresponding meaningless Z-matrix *coordinate* values can be omitted. More in general: missing coordinate values are set to zero (also for Cartesian coordinates input). Z-matrix values that are meaningless because they correspond to zero connection numbers are ignored, whatever their value is in the input file.

In a Z-matrix definition the three reference atoms, with respectively 3, 2, and 1 connection numbers equal to zero, do not have to be the first three in the input list. The program will scan the list for any atom that has 3

connection numbers zero, then for one that has only a bond length specification, etc. If the Z-matrix is not properly defined, for instance if more than one atom occurs with all three connection numbers equal to zero, or when not every atom is somehow connected to all others, the program will abort.

`F=Fragment`

Specifies that the atom belongs to a particular fragment. The fragment name must be of the form fragtype/n, where fragtype is the name of one of the types of fragments in the molecule. The integer n, after the slash, counts the individual fragments of that type. The numbering suffix /n is not required if there is only one fragment of that type.

When f=fragment is omitted altogether, the fragment type is taken to be the *atom type* that was specified earlier on the same line. (The numbering /n is then added automatically by the program, by counting the number of times that this single-atom fragment type occurs in the list of atoms).

`Mopac`

The MOPAC style input requires that the records in the data block have the following format:

atomtype distance idist angle iangle dihedral idehedral

The three internal coordinate values (distance, angle, dihedral) are each followed directly by the connection number.

Atom type is not identical to *chemical element*: an atom type is defined by all characteristics of the basic atom to which it in fact refers: the nuclear charge, the basis functions, the frozen core, the density functional and any other features that were applied in generating that basic atom.

As mentioned before, the point group symmetry specified in input with a Schönfliess type symbol puts restrictions on the orientation of the atomic system. Unless the input-specified symmetry equals the true symmetry of the nuclear frame (in which case adf will adjust the orientation of the molecule, if necessary), the user must take care of this by supplying the *Cartesian* coordinates (in the appropriate orientation). If a subgroup of the true nuclear symmetry is used and Z-matrix format is used for the coordinates, the program will place the atoms in the standard Z-matrix frame: first atom at the origin, second on the positive x-axis, third in the xy-plane with positive y-value.

Dummy atoms may be placed asymmetrically. If the atomic coordinates are input as Cartesians, any dummy atoms are irrelevant. Their coordinates will be printed but otherwise they are ignored.

Input items are generally case insensitive. Exceptions are the names of files and directories. Since (to be discussed below) the name of the fragment type as it is defined under atoms (explicitly with the f=option, or implicitly as the name of the atom type) might also directly indicate the fragment file, the specification of fragment types is in principle case-sensitive. Errors may occur if you are sloppy in this respect.

However, you must not give different fragment types names that differ only by case: at various places in the program fragment type names are compared in a case-insensitive way.

Mixed Cartesian and Z-matrix coordinates

The key ATOMS can also be used to supply coordinates in a format that gives the values for the cartesian coordinates *and* the connection matrix, which defines a Z-matrix.

```
ATOMS ZCart
 {N} Atom Coords {F=Fragment}
 ...
End
```

`ZCart`

Signals this particular format for the coordinates

`Coords`

As for Z-matrix input: three integers and three real values. The integers are the connection numbers that define the Z-matrix structure, but the reals are the *Cartesian* coordinates.

With ZCart input, the z-matrix is internally generated from the Cartesian coordinates and the connection numbers.

This feature is convenient when for instance Cartesian coordinates are easily available but you want to run a Geometry Optimization in *internal* coordinates, for which a Z-matrix structure is required.

The zcart option comes in handy also to satisfy symmetry-related orientation requirements when you basically wish to use Z-matrix coordinates.

With zcart input the program defines the *type* of coordinates in the input file as *Cartesian*. This is significant in Geometry Optimizations, where the optimization variables are by default taken as the input coordinate type.

Geometry Optimization

Geometry Optimizations in ADF is based on a quasi Newton approach [6-8], using the Hessian for computing changes in the geometry so as to make the gradients vanish. The Hessian itself is initialized (for instance based on a force field) and updated in the process of optimization.

Several subkeys in the geometry block can be used for control of the Geometry Optimization procedure and related strategy parameters.

```
GEOMETRY
 Optim {Cartesian / Internal / Delocal} {All / Selected}
 Iterations Niter {Niter2}
 Hessupd HessUpdate
 Converge {E=TolE} {Grad=tolG} {Rad=TolR} {Angle=tolA}
 Step {Rad=MaxRadStep} {Angle=MaxAngleStep} {TrustRadius=MaxRadius}
 DIIS {N=NVect} {CYC=Ncyc}
End
```

Optim

`Cartesian / Internal`

Cartesian or Zmatrix (equivalently: internal) specifies the type of coordinates in which the minimization is carried out. By default the coordinate type is applied that was used in the ATOMS key for the input of the (initial) atomic positions. (Cartesian if atoms were input in zcart format).

Cartesian optimization is allowed if the atoms were input in Z-matrix format, but no constraints (see the key GEOVAR) can then be used: *all* coordinates are optimized. An attempt to explicitly freeze variables may result in an error abort. Optimization in Z-matrix coordinates is not allowed if only Cartesian coordinates were supplied in atoms: the program does not construct a Z-matrix by itself. One should then use the zcart format: give Cartesian coordinates and supply the *structure* of the Z-matrix. Again, in this case you cannot use constraints.

`Delocal`

Optimization in delocalized coordinates (Delocal) can only be used in geometry optimizations or transition state searches. Work is in pogress to optimize this part of the code further. With delocalized coordinates you cannot use constraints and restraints, you can not supply the atoms in Z-matrix coordinates, and you can not set the initial hessian. See [173] for a definition of the delocalized coordinates.

`Selected`

Only those coordinates are optimized that are defined with the key GEOVAR.

`All`

(The default value) means that in principle all atomic coordinates will be varied. The key GEOVAR may modify this in the sense that some of the coordinates can be kept frozen or can be forced to remain equal to some other coordinates.

Iterations

`Niter`

The maximum number of geometry iterations allowed to locate the desired structure. The default is 30. This is a fairly large number. If the geometry has not converged (at least to a reasonable extent) within that many iterations you should sit down and consider the underlying cause rather than simply increase the allowed number of cycles and try again.

`Niter2`

An optional second parameter that plays only a role in a LinearTransit run, see the LT section. It must not be used in other runtypes.

Hessupd

`HessUpdate`

Specifies how the Hessian matrix is updated, using the gradient values of the current and the previous geometry.
Recognized values are:
(i) BFGS : Broyden-Fletcher-Goldfarb-Shanno
(ii) MS: Murtagh-Sargent
(iii) DFP : Davidon-Fletcher-Powell
(iv) FS : Fletcher switch
(v) HOSHINO : Hoshino
(vi) FARKAS : Farkas-Schlegel, Eq. (15) and (16) of Ref. [139]
(vii) FARKAS-BOFILL : Farkas-Schlegel-Bofill, Eq. (15) and (14) of Ref. [139]
default: BFGS.
Not that only BFGS and MS can be used in combination with delocalized coordinates, default: BFGS.

Converge

Convergence is monitored for three items: the energy, the Cartesian gradients and the estimated uncertainty in the (chosen type of optimization) coordinates. For the latter, lengths (Cartesian coordinates, bond-lengths) and angles (bond-, dihedral-) are considered separately.
Convergence criteria can be specified separately for each of these items:

`TolE`

The criterion for changes in the energy, in Hartrees. Default: 1e-3.

`TolG`

Applies to gradients, in Hartree/angstrom. Default: 1e-2.

`TolR`

Refers to changes in the Cartesian coordinates or bond lengths, depending on in what coordinates you optimize, in angstrom. Default: 1e-2.

`TolA`

Refers to changes in bond- and dihedral angles, in degrees. This is only meaningful if optimization takes place in Z-matrix coordinates. Default: 0.5 degree.
If only a numerical value is supplied as argument for *converge*, rather than a specification by name, it is considered to apply to the gradients (only). The other aspects (energy and coordinates) retain their default settings then.

Remarks:

1. Molecules may differ very much in the stiffness around the energy minimum. Application of standard convergence thresholds without second thought is therefore not recommended. Strict criteria may require a large number of steps, a loose threshold may yield geometries that are far from the minimum as regards atom-atom distances, bond-angles etc. even when the total energy of the molecule might be very close to the value at the minimum. It is good practice to consider first what the objectives of the calculation are. The default settings in ADF are intended to be reasonable for most applications but inevitably situations may arise where they are inadequate.

2. The numerical integration precision parameter accint (see the key INTEGRATION) should match the required level of convergence in gradients. Gradients are computed as a combination of various integrals that are evaluated by numerical integration in ADF. The integral values have a limited precision: roughly speaking the accint value is the number of decimal digits in the value of the integrals that are correct. As soon as the gradients, which are supposedly zero at the exact energy minimum, are of the order or 10**(-accint) they will, in worst cases, become arbitrary and any attempt to continue convergence may not really improve things. You may even find that, due to the numerical-integration noise, the geometries start moving around in a random fashion, while the gradients vary more or less arbitrarily. As a general rule: set the integration value higher (by *at least* 1.0) than the convergence level required for the gradients. Example: if gradients are to be converged to 1e-3, set integration 4.5 (implying: higher by 1.5 than the gradients convergence level).
3. The convergence threshold for the coordinates (TolL, TolA) is *not a* reliable measure for the precision of the final coordinates. Usually it yields a reasonable estimate (order of magnitude), but to get accurate results one should tighten the criterion on the gradients, rather than on the steps (coordinates). The reason for this is that the program-estimated uncertainty in the coordinates is related to the used Hessian, which is updated during the optimization. Quite often it stays rather far from an accurate representation of the true Hessian. This does usually not prevent the program from converging nicely, but it does imply a possibly incorrect calculation of the uncertainty in the coordinates.

Step

Controls that changes in geometry from one cycle to another are not too large:

`MaxRadStep`

An upper bound on changes in Cartesian coordinates or bond lengths, as the case may be. Default: 0.3 angstrom when optimization is carried out in internal coordinates, 0.15 angstrom for Cartesian optimizations.

`MaxAngleStep`

Similarly this option limits changes in bond angles and dihedral angles. Default: 10 degrees.
Input for MaxRadStep, MaxAngleStep is in angstrom and degrees respectively, independently of the units used for atomic coordinates input.

Note: Optimization of ring structures carried out in internal (z-matrix) coordinates is sometimes tricky due to the ill-defined last segment of the ring. When problems arise, try Cartesian optimization or consider using smaller limits on the steps (in particular the angles) so as to prevent the program from breaking the ring beyond repair.

`MaxRadius`

Can only be used in combination with delocalized coordinates. By default, the trust radius is set to 0.075 * √(num_atoms): 0.075 times the square root of the number of atoms. Using the key, the user can override this, setting a constant value. A conservative value is 0.2. A large system (eg 100 atoms) typically needs a larger trust radius (eg 0.8).

DIIS `N=NVect CYC=Ncyc`

Can only be used in combination with delocalized coordinates. NVect is the number of vectors used by the DIIS interpolation method. NCYC is the number of geometry cycles run before the DIIS starts to modify the geometry steps. Default DIIS is not used.

Transition State

A transition state (TS) search is very much like a minimization: the purpose is to find a stationary point on the energy surface, primarily by monitoring the energy gradients, which should vanish. The difference between a transition state and a (local) minimum is that at the transition state the Hessian has a negative eigenvalue.

Because of the similarities between a minimization and a TS search most subkeys in geometry are applicable in both cases, see the Geometry Optimization section. However, practice shows that transition states are much harder to compute than a minimum. For a large part this is due to the much stronger anharmonicities that usually occur near the ts, which threaten to invalidate the quasi-Newton methods to find the stationary point. For this reason it is good advice to be more cautious in the optimization strategy when approaching a Transition State and for some subkeys the default settings are indeed different from those for a simple optimization. In addition, certain additional aspects have to be addressed.

```
GEOMETRY
 TransitionState {Mode=Mode} {NegHess=NegHess}
end
```

`NegHess`

The number of negative eigenvalues that the Hessian should have at the saddle point. In the current release it is a rather meaningless key, which should retain its default value (1).

`Mode`

Controls the first step from the starting geometry towards the saddle point: it specifies in which direction the energy is to be *maximized* while the optimization coordinates will otherwise be varied so as to *minimize* the energy. A positive value means that the eigenvector #mode of the (initial) Hessian will be taken for the maximization direction. This means: put all Hessian eigenvalues in ascending order, ignoring those that correspond to impossible movements (rigid rotations and translations, symmetry breaking) and then take the eigenvector of #mode in the remaining list. A negative value for mode instructs the program to take the eigenvector that makes the largest change of the abs(mode)-th atomic coordinate (counting only the coordinates that are allowed to be changed independently, in order as they occur in the input list of coordinates under atoms).

Default: mode=1. Generally the program performs best with this default: it will simply concentrates on the mode with the lowest eigenvalue, which should of course finally be the path over the transition state

(negative eigenvalue!).

After the first geometry step, the subsequent steps will attempt to maximize along the eigenvector that resembles most (by overlap) the previous maximization direction until the Hessian is found to have a negative eigenvalue. At that point the program switches to that mode. As soon as the program has focused on the eigenvector with the lowest eigenvalue (mode=1) the overlap criterion to select the search direction is internally discarded and subsequently only the lowest eigenvector is taken. An input value mode=0 effectuates this immediately: the direction with the lowest eigenvalue will be the maximization direction for all iterations.

As mentioned before, the other subkeys have the same functionality as for minimizations, but different defaults or options may apply:

Hessupd

`HessianUpdate`

Different (fewer) options apply now:
(i) Powell: Powell
(ii) BFGS: Broyden-Fletcher-Goldfarb-Shanno
(iii) DFP: Davidon-Fletcher-Powell
(iv) BOFILL : Bofill, Eq. (13) and (14) of Ref. [139]
(v) MS: Murtagh-Sargent
default: Powell
Not that only POWELL, BFGS, BOFILL, and MS can be used in combination with delocalized coordinates, default: Powell.

Step

`MaxRadStep`

Default: 0.2 angstrom for Z-matrix optimization, 0.1 angstrom for Cartesian optimization.

`MaxAngleStep`

Default: 5 degrees.

Note: in Transition State searches precision is often much more critical than in minimizations. One should set the Numerical Integration precision at a fair value (4.5 at least). The *default* (i.e. automatic) value is 5.0 in a Transition State search.

Linear Transit

In a Linear Transit (LT) run you define a number of atomic coordinates (at least one) to be the LT *parameters*: these get an initial and a final value. The LT is defined as the simultaneous linear change of these parameters from their initial to their final values. This is carried out in a number of equidistant steps. The total number of LT *points* is specified on input. At each LT point the remaining atomic coordinates - those that are not LT parameters - may or may not be optimized: the (final) structure and energy at each LT point are computed. A Linear Transit (lt) run is therefore just a sequence of (related) constrained Geometry Optimizations.

The LT scan may be used for instance to sketch an approximate path over the transition states between reactants and products. From this a reasonable guess for the Transition State can be obtained which may serve as starting point for a true transition state search for instance.

Whenever a geometry subkey is applicable in a Geometry Optimization, it will apply in a Linear Transit run in each of the optimizations that are carried out at the distinct Linear Transit points, and the same default

values apply.

The runtype has to be specified. Additional specifications are optional.

```
GEOMETRY
 LinearTransit {NPoints}
end
```

NPoints

The number of LT points for which an optimization will be carried out
If no value is supplied the default takes effect: 5.

There are a few obvious differences between a single optimization and a LT run. Most important is that the coordinate(s) that describe the LT path, the LT *parameters*, cannot be optimized: at each of the LT points they are frozen. This implies that technically speaking at each LT point a *constrained* optimization is carried out. One of the consequences is that the atoms coordinate type - Cartesian or Z-matrix - must also be the optimization coordinate type. The LT parameters themselves must be defined with the key GEOVAR, see below.

It is possible to freeze *all* coordinates so that the LT run is similar to a sequence of Single Point runs. However, energy gradients will be computed at each step, so that more CPU time is spent at each LT point than for just a Single Point calculation.

The number of LT points by which the path is traced is defined by the npoints argument to the subkey LinearTransit. It is possible to execute only a subset of these points, usually with the purpose to complete the calculation by using the restart facility of ADF. In this way you can break down a very large calculation into several smaller ones, or have the opportunity to check how things have been going for the first few LT points before deciding whether a continuation is useful. This may be achieved of course by simply defining different start- and end-values for the LT parameters in a related series of calculations, but it is more comfortable to specify the complete path once and just execute parts of it at a time. This is accomplished by giving a *second* value to the iterations *s*ubkey in the geometry block.

```
...
iterations Niter Niter2
...
```

Niter

The first argument of the subkey iterations in the GEOMETRY block, controlling the maximum number of iterations allowed to reach convergence, applies now for each LT point separately.

Niter2

The second argument specifies the maximum number of LT *points* to calculate in this run. If omitted (default) the whole LT scan is completed. Doing only part of the scan may be combined with the restart feature, so that the remainder can be done in a continuation run. See the restart key.
A too large value of LT points is automatically adjusted: no more LT points are computed than required to complete the LT path as defined by the lineartransit subkey. A negative or zero value is not accepted and internally reset to one (1).

WARNING: if you use the QMMM functionality in combination with a Linear Transit, then only the coordinates of the true QM atoms can be used as LT parameters, no MM atoms must be involved in the LT parameter set.

Intrinsic Reaction Coordinate

The path of a chemical reaction can be traced from the Transition State to the products and/or to the reactants, using the Intrinsic Reaction Coordinate method (IRC) [9,10]. The starting coordinates should be a fair approximation of the Transition State. The final values at the endpoint(s) - reactants, products - are computed. The IRC path is defined as the steepest-descent path from the Transition State down to the local energy minimum. The energy profile is obtained as well as length and curvature properties of the path, providing the basic quantities for an analysis of the reaction path. Additional properties along the path (dipole moment, atomic charges) are computed.

Technically speaking the path is computed by taking small steps along the path meanwhile optimizing all atomic coordinates *orthogonal to it* so that, like in a Linear Transit run, a sequence of constrained optimizations is carried out. The total number of steps along the path is not known in advance. The maximum number of such steps can be set in input. If the path is not completed in the run, a Restart can be used to finish it. Each of the constrained optimizations in the run is treated as it would be in a Linear Transit run: convergence thresholds, maximum numbers of optimization iterations et cetera are set with subkeys in the geometry block.

You can set the IRC runtype by typing it in the geometry block

```
GEOMETRY
 IRC {Forward} {Backward} {Points=Points} {Step=Step} {StepMax=StepMax}
     {StepMin=StepMin} {Start=Start}
End
```

`IRC`

The runtype
IntrinsicReactionCoordinate would also be recognized.

`Forward, Backward`

Specifies execution of the two possible paths from the Transition State to the adjacent local minima. By default both are computed. If Forward is specified only, the other path is turned off and similarly for Backward. For the definition of which of the two directions down from the Transition State to an adjacent minimum is 'forward' see below.

`Points`

The maximum number of IRC points computed in the run, for both paths together and including the initial (TS) central point (as far as applicable). Default 100.

`Step`

The (initial) step length when proceeding from one IRC point to another along the path. The difference between two geometries, to which the step quantity applies, is measured in mass-weighted coordinates. The default value for step is 0.2 (amu)1/2 bohr. Larger steps reduce, in principle, the required number of IRC points from the transition state to the minimum, but usually at the expense of more optimization steps at each of the points so the net gain in computation time may not be very large, or even negative.
The default size is rather conservative and in many cases you may increase it to save a few steps. However, to some extent you can leave that to the program. When going from one point to the next, the program will increase or decrease the stepsize depending on whether or not the previous point to a large number of geometry cycles to converge. The adjusting algorithm also tends to be more cautious when the successive IRC points show more drastic changes in the atomic geometrical configuration. In all cases the IRC step sizes remain between pre-set maximum and minimum values, see the next items.

`StepMax`

The maximum step length that the program will select in the step-adjusting algorithm. Default: 1.0 or 10 times the initial step length, whichever is larger.

`StepMin`

The minimum step length that the program will select in the step-adjusting algorithm. Default: 0.03 or 0.3 times the initial step length, whichever is smaller.

`Start`

Defines how the initial direction of the path is chosen to move away from the Transition State. It does *not* imply whether the first step along this direction is taken positively or negatively. See for this aspect the section about Forward/Backward IRC paths.
The admissible values for start are:
Grad: compute the gradient and take that direction right from the start. Obviously, if we start perfectly at the Transition State this will be meaningless since the gradient vanishes there completely.
Read : the initial path direction is read in with the key IRCstart, see the section IRC Start Direction.
Hess n : the initial path coincides with the n-th Hessian eigenvector (ordered by ascending eigenvalues); *n* must be an integer in the appropriate range.
The default (omission of any start specification at all) is the first Hessian eigenvector, presumably corresponding to the path over the Transition State (negative Hessian eigenvalue!).

IRC start direction

As mentioned above, the IRC path is initialized by a first step away from the Transition State. If perfect information is available this should be along the unique Hessian eigenvector with a negative eigenvalue. Therefore, it is preferable to supply (with a restart file) a good approximation of the Hessian at the Transition State. This can be computed in a Frequencies run. In many cases the automatic internally generated (force field based) Hessian will not severely disturb the procedure and may only require a few more initial search steps for the right direction to take, while saving a potentially expensive Frequencies calculation.

If you decide to use a precalculated Hessian, then usually the approximate Hessian resulting from a Transition State run will be good enough. The latter approach is more attractive of course since the TS run will usually be done anyway, as a preliminary to the IRC run, while an additional Frequencies run would be very demanding. At the other hand, Transition State runs often require a preceding Frequencies run. In such case, the Frequencies result file may be used both for the TS run and for the IRC run. The fact that the Frequencies run may have been performed not at the exact TS may affect slightly the adequacy for using it as a start-up for the IRC run, but this is likely not significant.

In some case you may want to specify the initial direction of the IRC path explicitly. This is done as follows:

```
IRCSTART
 data
 data
 ...
end
```

`IRCstart`

A block-type key. The data in the data block are values for *all* atomic coordinates (Cartesian or Z-matrix, as the case may be) that are not frozen and not (by geovar) explicitly instructed to remain equal. All such coordinate data together define a direction vector in the space of all (free) coordinates, which then serves as the initial segment of the IRC path.

Note that only a direction vector is defined here: the *size* of the total vector plays no role.

Furthermore, the initial step may be in the positive or negative direction along the so-defined initial path, see the section Forward / Backward IRC paths.

Forward / Backward IRC paths

Obviously there are two IRC paths down the transition state: Forward and Backward. We would have liked to chemically define forward and backward by determining (in advance!) which of the endpoints is reactants and which products. This is not well doable in practice. Therefore we define the directions in terms of the initial path vector: select simply the atomic coordinate with the largest (absolute) change in the initial vector and define *Forward* as the direction in which this coordinate *increases* and *Backward* as the direction in which it *decreases*.

Climbing-Image Nudged Elastic Band

The reaction path can be found by simultaneous optimization of a number of replicas of the system in question starting from some rough approximation [159]. In the simplest case, implemented in ADF, the initial approximation is just a polynomial interpolation between initial and final states (see keyword geovar). The images are optimized not independently of each other but, in fact, forces on each image depend on its neighbors. At each step the forces parallel to the reaction path are eliminated and a so-called spring force is added which keeps the image in the middle between its neighbors. This does not let images slide to the initial or final reaction state and ensures that they are evenly distributed along the reaction path. There are also options to distribute images more densely near the transition state (energy-dependent spring force).

Below is the list of NEB options:

```
GEOMETRY
  CINEB {NumImages}
  {NEBSPRING  Nspring  Spring Spring2 Spower}
  {NEBOPT OptMethod}
  {NEBECONO}
  {NOCLIMB}
End
```

`CINEB`

The runtype. Nudged will also be recognized.

`NumImages`

The number of NEB images excluding initial and final stated. The default is 8.

`NEBSPRING Nspring Spring Spring2 Spower`

Nspring determines the type of spring used, which, in turn, determines which of the spring parameters are used:

1: constant spring, spring=Spring
2: exponential scaling, spring = Spring+Spring2*exp((dE-dEmax)**Spower)
3: power scaling, spring = Spring+Spring2*(dE/dEmax)**Spower
4: another exponential with different meaning of Spower, spring = Spring+Spring2*exp((dE-dEmax)*Spower)
5: another exponential scaling very close to #4, spring = Spring+Spring2*(2**(dE-dEmax))**Spower)

Units for Spring and Spring2 are Hartree/bohr. Default values, when NEBSPRING is not present in the input, are 1 for Nspring and 0.1 for Spring. If NEBSPRING is specified with Nspring 1 then the Spring parameter is required. If Nspring>1 is specified then also Spring2 and Spower are required.

`NEBOPT OptMethod`

Specifies the optimization procedure.

Since NEB is conceptually different from simple optimization, not all or not always options used in simple geometry optimization applicable to NEB. There are two modes optimization modes available for NEB: global (covering all images simultaneously) and local (that is, local to each image). Each method has its pro's and con's. The global method usually converges in fewer steps than local because its Hessian takes into account all degrees of freedom at once. On the other hand, the size of the matrix may become too large for moderate-size system, which might lead to problems (one dimension of the Hessian matrix may be as large as N(atoms)*3*N(images)).
There are two geometry update methods available for both global and local optimization: Quasi-Newton and Conjugate-Gradient. Quasi-Newton is the preferred method at all times. NEB optimization in Z-matrix coordinates is not available at this time.

OptMethod can take any of the following values:

GLOBALQN: global Quasi-Newton. Preferred (and default) method.
GLOBALCG: global Conjugate-Gradient.
QN: Local Quasi-Newton. This method uses the same optimizer as the standard geometry optimization while other methods use specially developed optimizers.
CG: local Conjugate-Gradient.

NEBECONO

(local optimization only) Requests that when at some point an image's geometry converges this image will not be recalculated in subsequent steps. This option can be used to speed up calculation in the end when some images have already converged. Please note though that even if an image has converged at one point it may become "un-converged" at a subsequent point due to increase in spring force (which is determined by position of the image with respect to its neighbors). This option is irrelevant in case of the global optimization because then the convergence state of a single image is not determined.

NOCLIMB

Switches off the climbing-image feature. This option is generally not recommended and exists for debugging and troubleshooting purposes.

Optimization: Special Features

geovar: constrained optimization, Linear Transit and NEB parameters

The block key GeoVar is used

- To put restrictions on the number of coordinates that are varied and
- To define Linear Transit or NEB parameters and assign them initial and final (and in case of NEB - also intermediate) values.

geovar can also be used to assign (initial) values to coordinates without other implications, but this feature is accidental.

In the input section of atomic coordinates (key ATOMS) identifiers (names) may be used rather than numerical values wherever coordinate values are expected: x, y, z in case of Cartesian coordinate input; r, q, f in case of internal coordinates. All such identifiers must then be specified under geovar and assigned a value.

```
GEOVAR
 Name Data
 ...
end
```

`Name`

An identifier that can be used in place of a numerical value for one or more of the atomic coordinate values under atoms.

`Data`

Either of the following three formats:

1 A single value simply assigns the value to the corresponding atomic coordinate(s).

2 Two or more values (separated by a delimiter) imply that the corresponding atomic coordinate is a Linear Transit or a Nudged Elastic Band parameter. For Linear Transit, only two values are allowed in which case they specify initial and final values of the LT path, respectively. In case of a NEB calculation one can provide more than just initial and final values to get a better initial approximation of the reaction path. It is generally recommended (and in some cases necessary) to use more values. Intermediate images will be obtained by polynomial interpolation of degree N-1, where N is the number of values.

3 A single value followed by a letter F assigns the value to the corresponding atomic coordinates *and* specifies that these coordinates are frozen: they will not be optimized.

As regards the optimization of coordinates other than the frozen ones and the LT or NEB parameters, the meaning and effect of the input under geovar depends on the subkey *optim* in the geometry block:

If selected has been set, optimizations are carried out only for the coordinates that are referred to under geovar (and that are not Linear Transit parameters or Frozen). All coordinates that were input as simple numerical data under atoms are kept frozen then.

Alternatively, if selected has *not* been set (: all, the default) all atomic coordinates are optimized (except the Linear Transit parameters and the explicitly frozen coordinates). In that case, each assignments under geovar other than to freeze the coordinate or to define it as a Linear Transit parameter simply assigns an initial value to the pertaining coordinates. In this respect it is not different from typing the numerical value directly in the atoms block, except for the next aspect. Please note that whereas during a linear transit run the LT parameters are *never* optimized, the NEB parameters specified in the *geovar* section are always optimized.

The same identifier may be used for two or more coordinates in atoms. If they are varied (i.e. if they are not frozen) they will forcibly be kept equal throughout the optimization so that they constitute only one degree of freedom. Don't use the same geovar variable for coordinates that belong to atoms of different chemical types or to different *types* of coordinates (an angle and a bond length for instance). It is not sensible to do so and it will very probably lead to an error abort or to stupid results.

It is allowed to put as atomic coordinate under atoms *minus* a geovar variable name, i.e. the name preceded directly by a minus sign (without a blank in between!). The coordinate will then be kept equal, but with opposite sign, to coordinates that are defined by the same variable without the minus sign. The initial (and final, in case of a LT or NEB run) value for that coordinate is the negative of the geovar value.

Constrained optimizations: coordinate types

Restricted optimizations are performed by freezing certain coordinates, by explicitly referring to one and the same geovar identifier for different coordinates, or by using the selected option. In addition, they are implicit in each Linear Transit or IRC run. All restricted optimizations demand that the *type* of optimization variables (Cartesian or Z-matrix) equal the type of coordinates used in atoms. zcart input under atoms is considered to be *Cartesian* in this respect.

If this is violated in a Linear Transit calculation the program will abort. If you apply the Frozen option under

geovar, while not using the same coordinate type for atoms as for optimization, an error will occur. If you refer to the same geovar identifier for distinct coordinates while the atoms and the optimization types of variables do not match, the program will continue and assume that you only have assigned the same *starting* values to the pertaining coordinates. No equality constraints will be in effect then during the optimization.

Constrained optimizations: linear combinations of internal coordinates

It is often desirable to carry out a geometry optimization in internal coordinates where two or more of the coordinates are required to maintain constant values relative to each other. The most simple case, where two internal coordinates a kept equal can be achieved by referencing both coordinates to a single variable in the GEOVAR block. Ensuring a different relationship, such as forcing one bond length in a molecule to be 0.5 Angstrom longer than another is more difficult to achieve. These kind of constraints can often be be managed through the creative use of dummy atoms but this is generally laborious and not always possible at all.

The CONSTRAINT keyword allows geometry optimizations with constraints defined by arbitrary linear combinations of internal coordinates to be performed quite straightforwardly. The keyword allows the linear combination to be constrained or used as part of a linear transit calculation with the constrained value being stepped as would a variable from the GEOVAR block.

```
CONSTRAINT
 Name1 Data1
 VAR11 Coef11
 VAR12 Coef12
 ...
 SUBEND
 Name2 Data2
 VAR21 Coef21
 VAR22 Coef22
 ...
 SUBEND
 ....
end
```

`Namex`

Identifier of the xth linear constraint.

`Datax`

Either of two formats
1) A single number, giving the value of the xth constraint
2) Two numbers, the first as in 1) and the second the final value in a linear transit calculation. The LINEARTRANSIT keyword must be present in the GEOMETRY block.

`Varxy`

Name of the yth variable in that is part of the xth constraint. Varxy must be defined in the GEOVAR block.

`Coeffxy`

Coefficient of Varxy in the linear combination defining the constraint. Thus:
`Coeff11*Var11 + Coeff12*Var12 ...= Data1`
The summation must be consistent with the initial values of the Varxy.

This procedure is only possible when the geometry is defined in terms of internal coordinates. Although the

program will not complain, it makes no sense to have linear combinations containing both bonds and angles of course.

The number of linear constraints must be less than or equal to the number of entries in the GEOVAR block. Only internal coordinates involving QM atoms can be included at this stage.

As a geometry optimization is run, the force acting on the linear constraints will be printed immediately after the forces on the internal coordinates. The constraint forces may be useful in the search for a transition state for instance.

Restrained optimizations

Up to now, the only way to constrain distances, angles or dihedral angles within a geometry optimization is by using a Z-matrix, and freezing that particular coordinate. With the key RESTRAINT it is possible to select **any** coordinate (distance, angle, dihedral), irrespective of the coordinates used, and restrain this coordinate.

Note the difference between *constrained* and *restrained* coordinates. At every step in the geometry optimization, the value of a *constrained* coordinate should match exactly a predefined fixed value. On the other hand, with *restraints*, a potential is added to the potential energy in order to satisfy the *restraint*, which means that the *restraint* does not have to be satisfied exactly. For example, one can start with a geometry in a geometry optimization run in which the restraint is not satisfied.

The RESTRAINT keyword allows geometry optimizations with restraints for the distance between two atoms, an angle defined by three atoms, (and) or a dihedral angle defined by four atoms:

```
RESTRAINT
 DIST Ia1 Ia2 Ra {[Aa] [Ba]}
 ANGLE Ib1 Ib2 Ib3 Rb {[Ab] [Bb]}
 DIHED Ic1 Ic2 Ic3 Ic4 Rc {[Ac] [Bc]}
end
```

DIST

When *DIST* is specified, the distance between atoms Ia1 and Ia2 is restrained to the value Ra. The atom numbers should be given in *Input order*; the value for the distance in Angstrom. The Aa and Ba values are mere technical values, that don't have to be specified (in fact, recommended not to change these values); the default values of 2.0 resp. 0.1 have been chosen on sensible grounds.

ANGLE

When *ANGLE* is specified, the angle between atoms Ib1, Ib2 and Ib3 (Ib1-Ib2-Ib3) is restrained to the value Rb. The atom numbers should be given in *Input order*; the value for the angle in degrees. The Aa and Ba values are mere technical values, that don't have to be specified (in fact, recommended not to change these values); the default values of 1.0 resp. 0.1 have been chosen on sensible grounds.

DIHED

When *DIHED* is specified, the dihedral angle between atoms Ic1, Ic2, Ic3 and Ic4 (Ic1-Ic2-Ic3-Ic4) is restrained to the value Rc. The atom numbers should be given in *Input order*; the value for the angle in degrees. The Aa and Ba values are mere technical values, that don't have to be specified (in fact, recommended not to change these values); the default values of 0.5 resp. 0.1 have been chosen on sensible grounds. The dihedral angle Ic1-Ic2-Ic3-Ic4 is defined in the same way as for the Z-matrix in ADF. The dihedral angle is projected onto the [0,2**TT**] interval, so there should be no difference between specifying -30° or 330°.

Symmetry versus constraints

The symmetry of the atomic system defined by the input Schönfliess symbol is preserved during optimization. If the input information (which coordinates are kept frozen and which are optimized) conflicts with the symmetry, the latter will prevail and an error exit may occur. In the program the geometric step is first computed according to the user-specified constraints (or restraints) and then symmetrized. This symmetrized geometry update is applied, regardless whether this results in frozen coordinates being changed.

Input specifications that are in conflict with the point group symmetry may lead to an error abort.

Z-matrix and symmetry

If the structure of the Z-matrix does not reflect the symmetry of the molecule *and* constraints are applied the program may encounter algorithmic problems to match all demands. As a result some of the *frozen* coordinates may be found to change. Usually these changes are very small. To cure this: build the Z-matrix in a symmetric way.

Symmetry in a Linear Transit

In a Linear Transit run it is imperative that the complete Linear Transit path as defined by the parameters conforms to the specified symmetry. If such is not the case, an error will occur or possibly the program will continue but not produce correct results. Note that when no symmetry is specified in input, the initial geometry defines the specified symmetry.

Summary of geovar, optim, and atoms

For *unconstrained* optimization: don't use geovar, apply *optim* if *Cartesian* optimization is required while

the data in the atoms block was in z-matrix format or when *z-matrix* optimization is required while the atoms input was in zcart format. Provide the atomic coordinates (atoms) directly as numerical data.

For optimizations where only very few coordinates are frozen: use geovar to set a few coordinates to frozen and/or to enforce equality of optimization coordinates whose values should remain equal. Don't use *optim*: the type of optimization coordinates - Cartesian or internal - must be identical to what is used in the atoms input part because you're using constraints now. In the atoms section, use identifiers for the frozen coordinates and for those that should satisfy equality conditions; use numerical input for all other (optimization) coordinates.

For very limited optimization: turn on the selected option with *optim* and assign with geovar initial values to the coordinates that you want to optimize. In the atoms input use identifiers for these coordinates. The numerical input coordinates are kept frozen automatically now.

Initial Hessian

In a Geometry Optimization (or Transition State search) the Hessian matrix - second derivatives of the energy with respect to changes in coordinates - is updated while the program steps around in an attempt to find the (local) energy minimum. The quality of the initial Hessian may have a considerable impact on the required number of steps to reach geometric convergence.

By default the initial Hessian is read from a restart file - see the key RESTART - or constructed from a force field [11] that is implemented in the program. In the latter case the user can modify the so-generated initial Hessian in four ways:

1. By setting all diagonal elements to some constant.
2. By defining *three* constants, one for distances (or Cartesian displacements, as the case may be), one for bond angles, and one for dihedral angles.
   All diagonal elements of the Hessian are adapted accordingly.
3. By supplying a list of diagonal values.
4. By giving diagonal-Hessian values for one or more specific coordinates.

For each element *i* for which a diagonal Hessian value Hii is supplied the off-diagonal elements Hij, (all j i≠j) are set to zero.

A combination of the above options is possible. The rules of how combinations are interpreted by the program are:

- The program first initializes the Hessian using the force field (or restart data).
- If a single constant (*1*) or three constants (*2*) are supplied, all diagonal elements are adjusted (and all off diagonal elements are set to zero).
- If a list of diagonal values is supplied (*3*), this overrides the first so many values of the diagonal. Such a list is not required to cover *all* diagonal elements.
  If the list is shorter than the dimension of the Hessian, i.e. the number of atomic coordinates, only the first so many elements will be adjusted.
- If any individual elements are supplied specifically (*4*), their values are replaced in the diagonal defined thus far.

All input values of the Hessian are in units of Hartree/bohr2 for Cartesian coordinates and bond lengths. Hartree/radian2 for bond angles and dihedral angles.

The first 3 options are controlled by the key HESSDIAG:

```
HESSDIAG {General}
{ List
end }
```

`HESSDIAG`

A *general* key: it has either an argument (General), or a data block (List). It is also possible to supply the argument *and* the data block, but this requires that the continuation symbol (&) is given after the argument, separated from the argument by at least one blank.

`General`

Must be either a single numerical value, or one or more named specifications of options, in the format optionname=value.

If a single numerical value is given, this value is assigned to all the options that are available. If the named-option format is applied, any named options that are not found get the value 1.0.
The options are: rad=radvalue to assign a value to all Hessian diagonal elements that refer to distance coordinates (bond length in case of z-matrix coordinates, Cartesian coordinates otherwise),

ang=angvalue to assign a value to all elements that refer to bond angles, and finally dih=dihvalue for dihedral angles.
ang and dih are not significant in Cartesian optimizations.

`List`

A list of numerical values, which may expand over any number of lines. If *n* numbers are supplied, they are assigned to the first *n* diagonal elements of the Hessian. The remaining diagonal elements, if any, are not effected. The maximum number of Hessian diagonal elements equals the number of atomic coordinates.

The force field derived initial Hessian can be printed for inspection. Type in input:

```
HESSTEST
```

ADF will construct and print the initial Hessian and then abort.

Hessian values for selected coordinates

The diagonal elements for selected free coordinates can be given if these free variables are named in the geovar block.

```
GEOVAR
 Varname Data H=HessValue
end
```

`Varname, Data`

The name of the variable and any data as discussed in the sections above: assignment of initial value, final value (in case of a Linear Transit run), or a Frozen specification.

`HessValue`

The value for the diagonal element of the Hessian associated with that variable. All atomic coordinates that are defined by this variable will get the HessValue as diagonal element in the initial force field. Specification of a HessValue for a frozen coordinate or a Linear Transit parameter is meaningless.

Frequencies

Harmonic frequencies can be computed in ADF either numerically or analytically. The frequencies are computed numerically by differentiation of energy gradients in slightly displaced geometries [12, 13]. The analytical second derivatives implementation in ADF is based on [208, 209, 210].

Analytical Frequencies

The frequencies are calculated analytically by specifying the **AnalyticalFreq** block keyword (see below for more details). The analytical frequencies are as accurate as the numerical frequencies for the same integration accuracy, but can be up to 3 to 5 times quicker to compute, depending on the molecule, integration grid parameters, and choice of basis set. The analytical frequencies are fully parallelised and linearly scaled.

Calculating the analytical frequencies requires the solution of the Coupled Perturbed Kohn-Sham (CPKS) equations, which is an iterative process. This part of the process is of order 3 x *number of atoms*, and is generally the main bottle neck in calculating the frequencies. (The immediate result of the solution of the CPKS equations is the U1 matrix, the components of which are closely related to the derivatives of the MO coefficients. One of the adjustable parameters in the input of an analytical frequencies calculation can be used to control the accuracy of the U1 matrix components.)

One disadvantage in calculating analytical frequencies is that the range of exchange-correlation functionals is limited. (This is because derivative formulas have to be derived for each exchange-correlation functional in

ADF, which is not an straight forward task). Here are the currently available functionals:

LDA: XONLY, VWN, STOLL
Exchange GGA: Becke88, OPTx, PBEx, rPBEx, revPBEx
Correlation GGA: LYP, Perdew86, PBEc
XC GGA shortcuts: BP86, PBE, RPBE, revPBE, BLYP, OLYP, OPBE

Any functional not mentioned above is not implemented, including PW91 and Hatree-Fock.

To calculate the frequencies analytically, include the block key word ANALYTICALFREQ. Subkeys are available, but in general, to calculate the frequencies, no subkeys are required, and including the following in your run file is sufficient:

```
AnalyticalFreq
End
```

Unlike the numerical frequencies, the analytical frequencies can be computed immediately after a geometry optimization by including both block keywords in the same input file.

```
Geometry
 ... Geometry optimization options here ...
End

AnalyticalFreq
End
```

A note of caution: during a geometry optimization the integration accuracy is set by default to 4, and so the resulting frequencies will also have this level of integration accuracy while it may be desirable to have frequencies computed with a higher accuracy. If accurate frequencies are required, then one should change the integration accuracy by using the INTEGRATION keyword.

The format for the AnalyticalFreq block key is:

```
AnalyticalFreq
    PRINT {eigs} {u1} {parts} {raw_freq}
    DEBUG {fit} {hessian} {b1} {densities} {numbers} {symmetry} {all}
    MAX_CPKS_ITERATIONS Niter
    CHECK_CPKS_FROM_ITERATION  N
    U1_ACCURACY  x
    NUC N1 N2 ... Nk
End
```

An explanation of the subkeys follow.

PRINT

This is primarly for debugging purposes. Choosing EIGS results in the print out of the MO eigenvectors, while U1 results in the print out of the U1 matrices. Except for small molecules this will result in a lot of data being output, and so they are not recommended. Choosing PARTS results in the print out of various sub-hessians that add up to give the final analytical hessian. RAW_FREQ gives the eignvalues of the initial force matrix, which are essentially the frequencies before rotational and translational degrees of freedom have been removed from the force matrix.

DEBUG

This is for debugging purposes. The choice FIT results in the print out of information related to the calculation of the fit coefficients and their derivatives. HESSIAN results in the printing out of many of the sub-Hessians that add ups to give the final Hessian. Many more Hessians are printed out with this option

that with the print parts subkey option (mentioned above). Choosing B1 gives data related to the frozen core orthogonalisation coefficients and their derivatives. DENSITIES gives the integrals and moments of various densities computed during the calculation of the frequencies. Including NUMBERS results in the print out of numbers of basis functions, fit functions etc, as well as various integer arrays that are crucial to the calculation of the analytical second derivatives. SYMMETRY results in symmetry information and symmetry matrices being printed out. ALL can be used to print out all debug information.

`MAX_CPKS_ITERATIONS Niter`

Calculating the analytical frequencies requires the solution of the Coupled Perturbed Kohn-Sham (CPKS) equations, which is an iterative process. For most systems tested so far, convergence to the required accuracy in the U1 matrix is achieved within Niter=20 iterations, which is the default. If convergence is not achieved (a warning will be printed in the output if this is the case) then this subkey can be used to increase the number of iterations, although convergence is not guaranteed. The user required accuracy of the U1 matrix, as well as the ADF integration accuracy, can effect the rates of convergence.

`CHECK_CPKS_FROM_ITERATION N`

Solution of the CPKS equations is an iterative process, and convergence is achieved if the difference between U1 matrix of successive iterations falls below a certain threshold. This key can be used to determine at which iteration the checking should start taking place. The default is 1.

`U1_ACCURACY x`

Solution of the CPKS equations is an iterative process, and convergence is achieved if the difference between U1 matrix of successive iterations falls below a certain threshold. This subkey can be used to set the threshold. The accuracy of the U1 will be 10**(-x). So, the higher the number the more accurate the U1 will be (similar to the integration accuracy parameter). The default is 4. While this parameter effects the accuracy of the frequencies, other factors also effect the accuracy of the frequencies, especially the ADF integration accuracy.

`NUC N1 N2 ... Nk`

By default, when calculating the frequencies analytically, the derivatives of the energy with respect to all nuclei are calculated. This gives a complete Hessian (second derivative) matrix, from which the vibrational frequencies of the molecule can be calculated. However, there may be certain cases where only derivatives with respect to a subset of all the nuclei are required. In this case it is a considerable saving in time if only a partial Hessian is calculated. With this subkey, a list of the nuclei for which the derivatives are required can be specified. However, the frequencies in this case are not the vibrational frequencies of the molecule, but may be used in guiding certain transition state searches.

Restarting Analytical Frequency jobs

Analytical frequency jobs can be restarted if a previous job did not finish. A restart can be done if the file TAPE21 has been saved without any corruption to the file. Upon doing a restart for analytical frequencies, the ADF program will check for the presence of an incomplete Hessian and/or for the presence of an incomplete U1 matrix, then attempt to figure out what more needs to be done.

The restart option may also be useful in combination with the atom selection option (i.e. by specifying a list of atoms following the keyword nuc in the analyticalfreq block keysee previous notes in this section). So for instance, you could calculate the partial Hessian for a subset of atoms of a molecule, and at a later time add another subset of atoms, or the rest of the atoms of the molecule to complete the Hessian.

Numerical Frequencies

Calculation of the numerical frequencies is specified using the FREQUENCIES keyword in the GEOMETRY

block. Most of the subkeys in the geometry block are meaningless for the calculation of frequencies. Indeed, a Frequencies calculation is not a variation on optimization, but rather a sequence of Single Point runs for the equilibrium geometry and a series of slightly different geometries. By comparison of the computed gradients the force constants and hence the frequencies are computed (in the harmonic approximation of the energy surface).

```
GEOMETRY
 Frequencies {Symm} {Allowed} {Numdif=Numdif} {Disrad=drad} {Disang=dang} {SCANALL} {NOSCAN}
 iterations Niter
end
```

Symm

This switch requests that frequencies are calculated in symmetric displacements. During such a calculation first symmetric atomic displacements are constructed. The number of such displacements in each irreducible representation corresponds to the number of frequencies with the corresponding symmetry. All displaced geometries within one representation have the same symmetry, which enables us to use it to speed up the computation significantly. Another advantage of having the same symmetry is that the numerical integration data can be reused efficiently (see SMOOTH option) thus reducing the level of numerical noise in gradients and force constant matrix. This is a new option and for now it only works with geometries specified in Cartesian coordinates. This option does not work correctly with a restart file. **This option does not work correctly when symmetry is explicitly specified in the input file.**

Allowed

Another advantage of the symmetric displacements is that only a subset of frequencies can be calculated. The ALLOWED option requests computation of only IR-visible frequencies. This option is only useful for symmetric molecules where it can be a big time-saver.

Numdif

Must have the value between 1 and 4 and specifies the type of numerical differentiation that is applied to compute the force constants from gradients in slightly displaced geometries: 1-, 2-, 3-, or 4-point numerical differentiation. In the case of 1-point differentiation the gradients of the displaced geometry are compared with the gradients at the input (equilibrium) geometry. In 2-point case both a negative and a positive displacement are applied, yielding much more accurate results but at the expense of more computations. This option is the default.
In certain cases the 3-point differentiation method gives better result than 2-point because it also takes gradients in the middle point into account. This is the case when geometry has not completely converged and the residual gradients are not quite close to zero. In this method, a formula is used that interpolates the second derivative (i.e. force constant) at the zero-force point. This way, the error due to a small deviation from the minimum geometry is decreased. The requirement is that the residual forces are small enough, more precisely, less that forces at displaced geometries (that is, using numdif=3 for arbitrary geometries is a bad idea).
When Numdif=4 is specified, force constants matrix will be computed by making two displacements in each direction, the standard (see drad, dang below) and twice as short. The force constant is then computed using the Romberg formula that reduces the higher-order and noise components: H(tot)=(4*H(dx/2)-H(dx))/3. Although this method requires twice as many single-point evaluations, one can probably get reliable results using lower integration accuracy, which might be faster than the default.

dang and drad

The displacements of the coordinates that will be varied. Dang applies to angles (bond and dihedral) in degrees and drad applies to Cartesian (x, y, z) coordinates and to bond lengths, in angstrom. Defaults: 1 degree and 0.01 angstrom.

Niter

In a calculation of frequencies it is the total number of (displaced) geometries for which gradients are computed. By default this is internally determined such that the calculation of frequencies can be completed. If you reduce it, the run will only partially build the matrix of force constants and a restart is required to complete the computation.

WARNING: you cannot combine a Frequencies calculation with the QMMM feature.

`SCANALL and NOSCAN`

ADF can scan some or all normal modes after a frequency calculation to verify the corresponding frequencies. By default, the normal modes corresponding all found imaginary frequencies are scanned. This can be switched off by specifying NOSCAN. Specifying SCANALL will tell ADF to scan along **all** normal modes. These options apply only to calculations in atomic displacements, that is when SYMM is not specified. These two options are mutually exclusive, the one specified last taking precedence.

Accuracy

Accuracy is a crucial aspect in the computation of frequencies, in particular for modes with low frequencies: the gradients at the geometries displaced along that mode will hardly change - analytically - from their equilibrium values, so numerical integration noise may easily affect the reliability of the computed *differences* in gradients. It is worthwhile to consider carefully the *size* of the displacements. At one hand they should be small in order to suppress the effect of higher order (anharmonic) terms in the energy surface around the minimum, at the other hand they should be large enough to get significant differences in gradients so that these are computed reliably.

High precision calculations where low frequency modes are involved may require high integration settings [14]. The default (i.e. automatic) value in a FREQUENCIES run is 6.0 (!). This may not be necessary in all cases, but it turns out to be required quite often in order to get accurate results. The calculation of frequencies by evaluating a series of displaced geometries, as it is implemented in ADF, is very time-consuming. This is even more so in view of the high (default) integration precision. This means that you should be prepared for long calculations.

Using 2-point differentiation rather than 1-point differentiation implies two-sided displacements of the atoms. This doubles the computational effort but in the so-computed force constants all anharmonic terms of *odd* order are eliminated. Since in general the lowest anharmonicity is third order this eliminates the first anharmonicity. Again, this is a feature directed primarily at obtaining highly accurate and reliable results.

The 3- and 4-point methods are intended to assist in special cases and as an extra check when the results obtained with the 2-point formula are not satisfactory. The 3-point formula should be used when residual forces after geometry optimization are between 0.01 and 0.0001 a.u./angstrom. In this case frequencies obtained with the 3-point formula are much closer to those that would be computed at the exact optimum geometry.

If a Frequencies calculation is carried out only to construct a good start-up Hessian for a TS search (see the restart key), accurate results are not crucial. The most important thing in such a situation is to get a fair guess for the negative eigenvalue and its associated mode, and to avoid spurious additional negative eigenvalues. We recommend to avoid the rather time-consuming standard Hessian-computing preparation run for a TS search and to lower the precision of the Frequencies run. A reasonable value should be 4.0.

Cartesian versus Z-matrix displacements

Cartesian displacements yield usually a higher accuracy than *Z-matrix* displacements because in the former case cancellation of numerical integration errors between the different geometries is (almost always) larger.

If Z-matrix coordinates are used as the displacement variables, then make sure that no bond angles of 180 (or zero) degrees are among them. They will very probably be treated incorrectly. If your molecule has such bond angles, use dummies to redefine the coordinates or use Cartesian displacements.

Frequencies and GEOVAR keyword

The use of the GEOVAR keyword in combination with a Frequencies run implies that constraints may be applied to the displacements, even if no coordinates are explicitly frozen. If different coordinates are connected to the same variable in the GEOVAR block, only combined displacements of the atoms will be allowed that correspond to a small change in the GEOVAR variable. For this reason, the combination of the GEOVAR and Frequencies keywords is to be handled with extreme caution. If no constraints are intended, it is recommended not to use the GEOVAR keyword, but to use DEFINE instead, or to specify the coordinates explicitly

Isotope Shifts of Vibrational Frequencies

To calculate isotopic shifts using ADF do the following:

- Calculate frequencies and save TAPE21 with a different name, say result.t21
- Modify the input file as follows:
    - add "RESTART result.t21" anywhere in the input file
    - create new fragment file with different mass
    - specify the fragment file in FRAGMENTS section
- Run ADF with the new input

Please note that if you change the fragment file for an atom that has symmetry-equivalent ones then the new fragment file will be applied to all of the atoms.
Example: first calculate the NH3 frequencies in the C(3v) symmetry and then change H to D. This will mean that one calculates the frequencies of a ND3 molecule and not of NH2D as one might want to do. If one wants to calculate the frequencies of NH2D one first has to do a calculation with lower symmetry, say C(s), to be able to change isotope of only one of the hydrogens.

Smoothing of Gradients

In ADF 2004.01 a method is implemented which is designed to smooth the gradient for small-ish perturbations in molecular geometry. This should help convergence in the last stages of a geometry optimization, and frequency calculations. We anticipate, for example, that it will be possible to perform frequency calculations with accint 4 with this option, rather than 5 or 6.

The reason for the smoothing is as follows: ADF generates integration points by dividing the 3D space up into Voronoy cells, and a spherical region around each atom. Unfortunately, the topology of the Voronoy cells is not always stable. The result is that in virtually every step in a geometry optimization the number of integration points changes. This can cause 'noise' in the gradient: even though the error in the gradient may not be excessively large, its magnitude and sign varies randomly with each change in geometry. This can cause the hessian (second derivative matrix) to be of poor quality. The new smoothing method is designed to make the error in the gradient vary systematically.

The way the smoothing works is to freeze the Voronoy cells in place from one step to the next whenever possible. The atoms are allowed to move (with their spherical regions) within these cells. Obviously after the atoms have been perturbed, the cells are no longer Voronoy cells of the molecular geometry, but there is nothing in the integration scheme that requires this.

By fixing the cells, we are able to regenerate the same integration points and weights each step. These points are shifted, and the weights adjusted, according to the atom's position in the cell. If an atom gets too close to the side of a cell, the freezing is relaxed, and the Voronoy cells recalculated. Another attempt to freeze the cells is then made at the next step. The smoothing can be made more effective if the DISHUL parameter in the INTEGRATION block key is increased, for example, to dishul=5.

This smoothing technique has been tested and found to considerably improve geometry optimization and frequency calculation results due to reduction of the numerical noise. As of ADF2005 smoothing is switched on for frequency calculations. It is off in all other cases by default. You can turn it on for any geometry run (e.g. geometry optimization, TS search, linear transit, etc) by using the 'smooth' subkey. You can use this in several ways. The first option is less dramatic:

```
GEOMETRY
 smooth freezecells
end
```

This option attempts to freeze the Voronoy cells between geometry steps, but does not reuse the points from the previous step. The points are instead recalculated, using the standard test integrals. Because the topology of the cells does not change, it is thought this may help somewhat, whilst still providing a rigorously tested integration grid.

The second option is more severe, but also more effective.

```
GEOMETRY
 smooth conservepoints
end
```

This option not only freezes the cells between steps, but also reuses the integration points of the previous step. It is recommended for frequency runs, as it should result in better gradient smoothing. The only disadvantage of this method is that there is no guarantee that the integral tests that ADF uses would be passed by the perturbed grid.

The third option is the most aggressive, but can be also most effective.

```
GEOMETRY
 smooth aggressive
end
```

This option not only freezes the cells between steps, reuses the integration points of the previous step, but also ignores some checks, which might lead to extra cells. By default, this option is on during frequency calculations.

The smoothing should be particularly effective for frequency calculations of molecules with no symmetry. In theory one should be able to rerun old frequency calculations with lower accint, for example, and still accurately reproduce the frequencies.

The method should help in the last stages of geometry optimizations, where the geometry is almost converged. In theory, you should now be able to use much lower gradient tolerances than were previously possible. It should also be possible to converge optimizations with lower accint than previously possible. accint 4 should suffice, rather than accint 6.

Geometry optimizations in which the molecule almost reaches convergence, but then continuously takes small steps around the minimum, should benefit greatly from the gradient smoothing.

Fragments

Fragment files

The TAPE21 result files from the ADF computations on the fragments that constitute a molecule completely characterize these fragments. The fragment TAPE21 files must be attached as *fragment files*. This is achieved with the key FRAGMENTS. See also the next section for the relation between Atom type, Fragment type and Fragment file names.

```
FRAGMENTS {Directory}
 FragType FragFile
 FragType FragFile
 ...
end
```

`FragType`

One of the fragment *types* defined under atoms, either explicitly (f=fragtype/n) or implicitly (fragment type=atom type, if the f= option is not used).

`FragFile`

The fragment file: the standard TAPE21 result file from the computation of that fragment. The file name must contain the complete path relative to Directory (the argument of the key). By default, when no Directory is specified, this is the local directory where the job runs. You may therefore omit the directory and give simple (local) file names if all the files are present in the working directory of the job.

Obviously, FragFile is case sensitive. However, FragType is also treated as case sensitive; see also the ATOMS key discussion (f= option). The reason is that there are shortcuts possible to the effect that the FragType name (in the atoms block) is immediately interpreted as the name of the fragment file.

The key FRAGMENTS may be used any number of times in the input file. This is convenient if you employ a sizeable number of fragment files, with subsets located in different directories. You can then use the key separately for each directory, to avoid typing long path names for all the files. Fragtypes that occur in the fragments block(s), but that are not referred by atoms are ignored. No fragment files must be specified for dummy atoms (xx).

It is allowed to use one and the same fragment file for different fragment types. Example:

```
ATOMS
 C.1 x1 y1 z1
 C.2 x2 y2 z2
 ...
end
fragments
 C.1 TAPE21.c
 C.2 TAPE21.c
 ...
end
```

Two different atom types (and fragment types) C.1 and C.2 are defined. The properties of the two fragment types are now identical since they are characterized by the same fragment file, but from the program's point of view they are different and can therefore not be symmetry equivalent.

The reason you may want to specify different atom types will usually be related to analysis, in particular symmetry aspects. If you know in advance that the two atom types are not symmetry equivalent, or more generally, that they play a rather different role in the molecule, it can enhance clarity of printed output to assign different atom type names to them. However, see the notes below.

A fragment file must not be the result file of a spin-unrestricted calculation. When you try to use such a fragment file, the program will detect it and abort with an error message. If you want to analyze a molecule in terms of unrestricted fragments, you should use restricted fragment files and apply the key FRAGOCCUPATIONS.

Suppose that you have done a calculation on a molecule *mol*, in which you have defined two different atom types for atoms of the same chemical element. Suppose furthermore, that you want to use that molecule now as a fragment in a new calculation.

You list under atoms all atoms of the molecule and you specify which atoms belong to the various fragments, among which the molecular fragment *mol*. The program will then have a problem deciding which atoms in your system are associated with the different atom types in the fragment. Normally, ADF analyzes this by comparing the chemical elements. That is not sufficient here because one chemical element corresponds with more than one type of atom in the *mol fragment* type. In such a case it is imperative to use *the same atom type names* in your new calculation as you used in the generation of the fragment. These names are stored in the fragment file, and they are printed in the output file of the calculation of *mol*.

The names of three items may be related to each other, depending on how you specify input: the *atom type*, the *fragment type*, and the *fragment file*.

The atom type is defined in the data block to atoms.

The fragment type is defined also in the data block to atoms: with the f= option. For records in the data block that don't have the f= option, the fragment type name is by definition identical to the atom type name.

The fragment file is defined in the data block to fragments, each record consisting of a fragment *type* name, followed by the fragment *file*. If a fragment type is not listed in the data block to fragments, so that no fragment file name is specified, the fragment *file* is by definition identical to the fragment *type* name.

QM/MM

ADF supports the QM/MM method to handle large systems or environment effects by treating only part of the atoms quantum-mechanically and the other ones by molecular mechanics. Use of this feature is invoked by the QM/MM keyword (block type). The functionality and all details of the keyword, involving quite a few options and aspects, are described in the separate QM/MM manual.

See also the pdb2adf utility, described in detail in the Utilities document, which transforms a PDB file into an ADF input file, for use with QM/MM.

Density Functional

Exchange Correlation Functionals

The Density Functional, also called the exchange-and-correlation (XC) functional, consists of an LDA, a GGA part, and possibly a Hartree-Fock exchange part (hybrids). LDA stands for the Local Density Approximation, which implies that the XC functional in each point in space depends only on the (spin) density in that same point. GGA stands for Generalized Gradient Approximation and is an addition to the LDA part, by including terms that depend on derivatives of the density. A hybrid GGA (for example B3LYP) stands for some combination of a standard GGA with a part of Hartree Fock exchange. For these terms ADF supports a large number of the formulas advocated in the literature. For post-SCF energies only, ADF supports also various meta-GGA functionals and more hybrid functionals.

Recently the Perdew-Zunger self-interaction correction (SIC) was implemented [47-49] self-consistently using

the Krieger-Li-Iafrate approximation to the optimized effective potential, and the Vosko-Wilk-Nusair (VWN) functional or gradient corrected density functionals. This approach is found to improve several properties, which are sometimes difficult to describe with standard DFT techniques, like for example some 'problematic' NMR chemical shifts, or some 'difficult' reaction barriers.

Finally, several asymptotically correct XC potentials have been implemented in ADF, such as the (now somewhat outdated) LB94 potential [15], the gradient-regulated asymptotic correction (GRAC) [16], and the statistical average of orbital potentials (SAOP) [17]. These can currently be used only for response property calculations, not for geometry optimizations. For spectroscopic properties, they usually give results superior to those obtained with LDA or GGA potentials, (see Ref.[18] for applications to (hyper)polarizabilities Cauchy coefficients, etc. of small molecules). This is particularly true if the molecule is small and the (high-lying) virtual orbitals are important for the property under study.

Recently it was also shown that, simply using the orbital energies of the occupied Kohn-Sham orbitals of a SAOP calculation, quite good agreement with experiment vertical ionization potentials is obtained. This is true not only for the HOMO orbital energy, which should be identical to (minus) the experimental ionization potential with the exact XC potential, but also for lower-lying occupied orbital energies. The agreement becomes worse for deep-lying core orbital energies. A theoretical explanation and practical results are given in Ref. [19].

In principle you may specify different functionals to be used for the *potential,* which determines the self-consistent charge density, and for the *energy* expression that is used to evaluate the (XC part of the) energy of the charge density. To be consistent, one should generally apply the same functional to evaluate the potential and energy respectively. Two reasons, however, may lead one to do otherwise:

- The evaluation of the GGA part in the *potential* is rather time-consuming.
  The effect of the GGA term in the potential on the self-consistent charge density is often not very large.
  From the point of view of computational efficiency it may, therefore, be attractive to solve the SCF equations at the LDA level (i.e. not including GGA terms in the potential), and
  to apply the full expression, including GGA terms, to the energy evaluation *a posteriori*: post-SCF.
- A particular XC functional may have only an implementation for the potential, but not for the energy (or vice versa). This is a rather special case, intended primarily for fundamental research of Density Functional Theory, rather than for run-of-the-mill production runs.

The key that controls the Density Functional is XC, with sub keys *LDA* and *GGA* (or equivalently: *gradients*) to define the LDA and GGA parts of the functional, and *MODEL* in case one of the special 'model' XC potentials is required in stead of LDA or GGA. All subkeys are optional (need not be used) and may occur twice in the data block: if one wants to specify different functionals for potential and energy evaluations respectively, see above.

```
XC
 {LDA {Apply} LDA {Stoll}}
 {GGA {Apply} GGA}
 {Model MODELPOT [IP]}
 {HARTREEFOCK}
 {HYBRID hybrid}
end
```

`Apply`

States whether the functional defined on the pertaining line will be used self-consistently (in the SCF-potential), or only post-SCF, i.e. to evaluate the XC energy corresponding to the charge density.
The value of apply must be SCF or Energy.
A value postSCF will also be accepted and is equivalent to Energy.
A value Potential will also be accepted and is equivalent to SCF.
For each record separately the default (if no Apply value is given in that record) is SCF.

For each of the two terms (LDA, GGA) in the functional: if no record with Energy specification is found in the data block, the evaluation of the XC energy will use the same functional as is applied for the potential.

`LDA`

Defines the LDA part of the XC functional and can be any of the following:
`Xonly`: The pure-exchange electron gas formula. Technically this is identical to the Xalpha form (see next) with a value 2/3 for the X-alpha parameter.
`Xalpha`: the scaled (parameterized) exchange-only formula. When this option is used you may (optionally) specify the X-alpha *parameter* by typing a numerical value after the string Xalpha (separated by a blank). If omitted this parameter takes the default value 0.7
`VWN`: the parameterization of electron gas data given by Vosko, Wilk and Nusair (ref [20], formula version V). Among the available LDA options this is the more advanced one, including correlation effects to a fair extent.

`Stoll`

For the VWN or GL variety of the LDA form you may include Stoll's correction [21] by typing Stoll on the same line, after the main LDA specification. You must not use Stoll's correction in combination with the Xonly or the Xalpha form for the Local Density functional.

`GGA`

Specifies the GGA part of the XC Functional, in earlier times often called the 'non-local' correction to the LDA part of the density functional. It uses derivatives (gradients) of the charge density. Separate choices can be made for the GGA exchange correction and the GGA correlation correction respectively. Both specifications must be typed (if at all) on the same line, after the GGA subkey.

For the exchange part the options are:
`Becke`: the gradient correction proposed in 1988 by Becke [22].
`PW86x`: the correction advocated in 1986 by Perdew-Wang [23].
`PW91x`: the exchange correction proposed in 1991 by Perdew-Wang [24]
`mPWx`: the modified PW91 exchange correction proposed in 1998 by Adamo-Barone [25]
`PBEx`: the exchange correction proposed in 1996 by Perdew-Burke-Ernzerhof [26]
`RPBEx`: the revised PBE exchange correction proposed in 1999 by Hammer-Hansen-Norskov [27]
`revPBEx`: the revised PBE exchange correction proposed in 1998 by Zhang-Wang [28]
`mPBEx`: the modified PBE exchange correction proposed in 2002 by Adamo-Barone [174]
`OPTX`: the OPTX exchange correction proposed in 2001 by Handy-Cohen [29]

For the correlation part the options are:
`Perdew`: the correlation term presented in 1986 by Perdew [30].
`PBEc`: the correlation term presented in 1996 by Perdew-Burke-Ernzerhof [26] .
`PW91c`: the correlation correction of Perdew-Wang (1991), see [24].
`LYP`: the Lee-Yang-Parr 1988 correlation correction [31-33].

Some GGA options define the exchange and correlation parts in one stroke. These are:
`BP86`: this is equivalent to Becke + Perdew together.
`PW91`: this is equivalent to pw91x + pw91c together.
`mPW`: this is equivalent to mPWx + pw91c together.
`PBE`: this is equivalent to PBEx + PBEc together
`RPBE`: this is equivalent to RPBEx + PBEc together
`revPBE`: this is equivalent to revPBEx + PBEc together
`mPBE`: this is equivalent to mPBEx + PBEc together
`BLYP`: this is equivalent to Becke (exchange) + LYP (correlation).
`OLYP`: this is equivalent to OPTX (exchange) + LYP (correlation).
`OPBE`: this is equivalent to OPTX (exchange) + PBEc (correlation) [175].
`XLYP`: this is equivalent to XLYPx [172] (exchange, not available separately from LYP) + LYP (correlation).

LB94: this refers to the XC functional of Van Leeuwen and Baerends [15].
KT1: this refers to the KT1 functional of Keal and Tozer [171].
KT2: this refers to the KT2 functional of Keal and Tozer [171].

The string GGA must contain not more than one of the exchange options and not more than one of the correlation options. If options are applied for both they must be separated by a blank or a comma.

MODEL

Specifies that one of the less common XC potentials should be used during the SCF. These potentials specify both the exchange and the correlation part. No LDA, GGA, HARTREEFOCK, or HYBRID key should be used in combination with these keys. It is also not advised to use any energy analysis in combination with these potentials. For energy analysis we recommend to use one of the GGA potentials. It is currently not possible to do a Create run with these potentials. It is possible to do a one atom regular ADF calculation with these potentials though, using a regular TAPE21 file from an LDA or GGA potential as input.

LB94: this refers to the XC functional of Van Leeuwen and Baerends [15]. There are no separate entries for the Exchange and Correlation parts respectively of LB94. Usually the GRACLB or SAOP potentials give results superior to LB94.

GRACLB: the gradient-regulated asymptotic correction, which in the outer region closely resembles the LB94 potential [16]. It requires a further argument: the ionization potential [IP] of the molecule, in hartree units. This should be estimated or obtained externally, or calculated in advance from two GGA total energy calculations.

SAOP: the statistical average of orbital potentials [17]. It can be used for all electron calculations only. It will be expensive for large molecules, but requires no further parameter input.

IP: should be supplied only if GRACLB is specified.

HARTREEFOCK

Specifies that the Hartree-Fock exchange should be used during the SCF. No LDA, GGA, MODEL, or HYBRID key should be used in combination with this key. Note that at the moment Hartree-Fock exchange can not be used in combination with frozen cores. The implementation in ADF of the calculation of exact exchange (Hartree Fock exchange), which is also needed for the hybrid functionals, is based on work by Watson et al., Ref. [138]. In ADF Hartree-Fock exchange only make sense if the calculation is performed with completely filled orbitals (ROHF is not implemented in ADF, only UHF). Note that the DEPENDENCY key is switched on for Hartree-Fock exchange in order to circumvent numerical problems, see also the ADDDIFFUSEFIT key.

HYBRID

Specifies that a hybrid functional should be used during the SCF. No LDA, GGA, MODEL, or HARTREEFOCK key should be used in combination with this key. Note that at the moment hybrid functionals can not be used in combination with frozen cores. In ADF the hybrid potential only make sense if the calculation is performed with completely filled orbitals (ROHF is not implemented in ADF, only UHF). Note that the DEPENDENCY key is switched on for hybrid functionals in order to circumvent numerical problems, see also the ADDDIFFUSEFIT key.

The hybrid can be one of the following:
B3LYP: ADF uses VWN5 in B3LYP. functional (20% Hartree-Fock exchange) by Stephens-Devlin-Chablowski-Frisch [176].
B3LYP*: modified B3LYP functional (15% Hartree-Fock exchange) by Reiher-Salomon-Hess [177].
B1LYP: functional (25% Hartree-Fock exchange) by Adamo-Barone [178].
KMLYP: functional (55.7% Hartree-Fock exchange) by Kang-Musgrave [179].
O3LYP: functional (12% Hartree-Fock exchange) by Cohen-Handy [180].
X3LYP: functional (21.8% Hartree-Fock exchange) by Xu-Goddard [172].

BHandH: 50% Hartree-Fock exchange, 50% LDA exchange, and 100% LYP correlation.
BHandHLYP: 50% Hartree-Fock exchange, 50% LDA exchange, 50% Becke88 exchange, and 100% LYP correlation.
B1PW91: functional by (25% Hartree-Fock exchange) Adamo-Barone [178].
mPW1PW: functional (42.8% Hartree-Fock exchange) by Adamo-Barone [25].
mPW1K: functional (25% Hartree-Fock exchange) by Lynch-Fast-Harris-Truhlar [181].
PBE0: functional (25% Hartree-Fock exchange) by Ernzerhof-Scuseria [211] and by Adamo-Barone [212], hybrid form of PBE.
OPBE0: functional (25% Hartree-Fock exchange) by Swart-Ehlers-Lammertsma [175], hybrid form of OPBE.

Defaults and special cases

If the XC key is not used, the program will apply only the Local Density Approximation (no GGA terms). The chosen LDA form is then VWN.

If only a GGA part is specified, omitting the *LDA* sub key, the LDA part defaults to VWN, except when the LYP correlation correction is used: in that case the LDA default is Xonly: pure exchange.

The reason for this is that the LYP formulas assume the pure-exchange LDA form, while for instance the Perdew-86 correlation correction is a correction to a *correlated* LDA form. The precise form of this correlated LDA form assumed in the Perdew-86 correlation correction is not available as an option in ADF but the VWN formulas are fairly close to it.

Be aware that typing only the sub key *LDA*, without an argument, will activate the VWN form (also if LYP is specified in the GGA part).

Model potentials

The LB94, GRAC, and SAOP functionals have only a SCF (=Potential) implementation, but no Energy counterpart. Therefore, they must not be used together with the Energy specification for Apply. If LB94 or GRAC is used for the Potential (SCF), the gga energy expression defaults to Becke (exchange part) + Perdew (correlation). For SAOP, the energy functional is PW91. This can be overruled by selecting another choice in the 'gga Energy ...' specification. However, it is recommendable to use a GGA for the XC potential if the main interest is in energies.

The LB94, GRAC, and SAOP forms are density functionals specifically designed to get the correct asymptotic behavior. This yields much better energies for the highest occupied molecular orbital (HOMO) and better excitation energies in a calculation of response properties (Time Dependent DFT). Energies for lower lying orbitals (sub-valence) should improve as well (in case of GRAC and SAOP, but not LB94). The energy expression underlying the LB94 functional is very inaccurate. This does not affect the response properties but it does imply that the energy and its derivatives (gradients) should not be used because lb94-optimized geometries will be wrong, see for instance [34]. The application of the LB94 functional in a runtype that involves the computation of energy gradients is disabled in ADF. You can override this internal check with the key ALLOW.

In case of a GRACLB calculation, the user should be aware that the potential in the outer region is shifted up with respect to the usual level. In other words, the XC potential does not tend to zero in the outer region in this case. The size of the shift is the difference between the HOMO orbital energy and the IP given as input. In order to compare to regular GGA orbital energies, it is advisable to subtract this amount from all orbital energies. Of course, orbital energy differences, which enter excitation energies, are not affected by this shift in the potential.

The LB94, SAOP, and GRAC potentials cannot be used in a Create run (due to an implementation limitation in the code). If you need the energy difference of a molecule with respect to LB94-atoms, you have to run the single-atom calculations with LB94 separately, using the same non-LB94 Create atoms as fragments as you did for the whole molecule. This will give you the required energy corrections.

Hartree-Fock and hybrid potentials

Hartree-Fock and the hybrid potentials can not or should not be used in combination with geometry optimization, TS, IRC, LT, frequencies, NMR chemical shift, spin-spin coupling, time dependent DFT, EPR g-tensor, frozen cores.

Numerical problems have been found with the present implementation of Hartree-Fock or hybrids during the SCF, especially if the molecule has symmetry NOSYM and a basis set TZP or larger is used. Workaround is to use always the DEPENDENCY key with rather strict criteria for the basis set dependence, namely bas=4e-3. In the ADF2006.01 the DEPENDENCY key is automatically switched on in the case of a Hartree-Fock or a hybrid potential. The result of the DEPENDENCY key is that linear dependence of the basis set is reduced by removing linear combinations that correspond with eigenvalues in the virtual SFOs overlap matrix, which are smaller than, in this case, 4e-3. Note that this is a rather large value, such that it will have an effect on the bonding energy. For DZ and DZP basis sets this value will normally not result in reduction of the virtual space. However, for TZP, TZ2P, QZ4P and larger this will often result in reduction of the basis set, which will have an effect on the accuracy of the bonding energy. In these cases one could try a smaller value than 4e-3, but be aware that numerical problems may occur. If the molecule has symmetry the numerical problems are reduced.

The origin of this problem is that for an accurate description of Hartree-Fock exchange one needs more (diffuse) fit functions in the fit procedure which is used in ADF, which uses only fit fuctions on the two centers of the two STOs. One can get more diffuse fit functions if one adds in the Create run of an atom the key:

```
AddDiffuseFit
```

If the BASIS key is used one can also add this key in the molecular calculation (the scripts in ADF will then automatically add this in the Create runs of the atoms). If one adds this key preliminary results indicate that one can lower the value for the dependency key to bas=1e-4. Such a low value for the dependency key normally means that the basis set is not reduced for basis sets of TZP or TZ2P quality.

Meta-GGA and hybrid energy functionals

In the ADF2004.01 version, several GGA [26-28,35-39], meta-GGA [40-46], hybrid GGA (for example B3LYP), and hybrid meta-GGA energy XC functionals have been implemented that have been shown to give good results for energies. The present implementation enables only energies to be calculated with most these functionals. Not all XC potentials have been implemented, so that for example geometry optimizations or property calculations cannot be performed with these functionals. For some GGA and hybrid functionals the XC potential is implemented.

The implementation in ADF of the calculation of exact exchange (Hartree Fock exchange), which is needed for the hybrid functionals, is based on work by Watson et al., Ref. [138]. The difference with their method is the way in which ADF the orbital densities are fitted.

The calculation of a large, prespecified list of LDA, GGA, and meta-GGA energy functionals is invoked by specifying

```
METAGGA
```

as a separate keyword. The hybrid GGA and hybrid meta-GGA energy functionals are calculated if in addition to the METAGGA key, the key

```
HARTREEFOCK
```

is included.

The keys METAGGA and HARTREEFOCK can be used in combination with any XC potential. Note that at the moment hybrid functionals can not be used in combination with frozen cores. Some METAGGA functionals will also give wrong results if used in combination with frozen cores. One should include the HARTREEFOCK keyword also in the create runs of the atoms. In ADF the hybrid energies only make sense if the calculation is performed with completely filled orbitals (ROHF is not implemented in ADF, only UHF).

For comparison with previous results instead of the key HARTREEFOCK one can also use the key HFEXCHANGE:

```
HFEXCHANGE
```

This key is now obsolete. The difference with the HARTREEFOCK key is the way in which the orbital densities are fitted. The key HFEXCHANGE can not be used in combination with frozen cores or spin-orbit coupling.

The Examples document describes an application to the OH molecule for the METAGGA option. More output, on the total XC energy of the system, can be obtained by specifying

```
PRINT METAGGA
```

This latter option is intended for debugging purposes mainly and is not recommended for general use.

The implementation calculates the total XC energy for a system and writes it to a file. This is always done in Create runs. If the basic fragments are atoms, the keyword

```
ENERGYFRAG
ATOM [filename]
ATOM [filename]
 ... ...
END
```

specifies that different atomic fragment files are to be used in the meta-GGA energy analysis than the regular atomic fragment files from the create runs. This keyword cannot be used for molecular fragment files. In order to compare meta-GGA energy differences between molecular fragments and the total molecule, results from the various calculations need to be combined by hand.

In such situations, it is advisable to use a somewhat higher integration accuracy than one would normally do, at least for the smaller fragments, as there is no error cancellation as in a regular ADF bond energy analysis.

A general comment is that some functionals show a more stable behavior than others (at least in our current implementation). In general, the functionals which are dependent on the Laplacian of the density may display a large variation with respect to basis set changes or different numerical integration accuracy. For this reason we currently recommend FT97 in favor of FT98. Similarly, the results with the BmTau1 functional should still be carefully checked. In our test calculations on the G2 set of molecules, the VS98 showed best performance, both for the average error and for the maximum error. The G2 set consists only of small molecules with elements up to Cl. The relative performance for transition metals and heavy elements is unknown and may well be very different from the ordering for the G2 set.

Self-Interaction Correction

In the ADF2004.01 version the Perdew-Zunger (PZ) self-interaction energy correction (SIC) with the Krieger-Li-Iafrate (KLI) approximation to the self-interaction corrected optimized effective potential (OEP) is implemented[47-49]. The block key SICOEP should be used.

Note: The existing auxiliary fitting sets, employed in ADF, were optimized for the calculation of the Coulomb potential of the total electron density. These standard fitting sets are quite flexible in the valence region, but do not include functions of high angular momentum in the inner regions. It was found (Ref [47]) that self-consistent SIC calculations with the standard fitting sets result in very poor SCF convergence, and large fit incompleteness corrections, particularly for the orbitals with substantial p and d contributions. Once the auxiliary fit sets are augmented with additional functions of high angular momentum, and reoptimized, the SCF convergence problems largely disappear. The reason is that the fit set must be able to approximate densities of each individual localized orbital, both in direction, and shell structure.

The key

```
SINGULARFIT FAST
```

activates fast(er) treatment of linearly dependent fits. It precomputes and stores singular value decomposition of fit overlap integrals, making fitting itself faster. Because SIC needs to fit a lot of densities on each iteration, it can improve performance by an order of magnitude. Use of 'SINGULARFIT FRUGAL' requests the old treatment of linearly-dependent fits, recomputing SVD parameters on each iteration. This is always slower than 'SINGULARFIT FAST', but uses less disk space.

The localization transformation is essential for obtaining realistic total and atomization energies. For example, without localization, SIC-Vosko-Wilk-Nusair (VWN) predicts molecular oxygen to be unstable with respect to the dissociation to two neutral oxygen atoms. At the same time, when using the localized orbitals, SIC-VWN compares favorably to both VWN itself, and to sophisticated gradient-corrected functionals. In ADF the Foster-Boys localization procedure is used.

Remarks: Fractional occupation are treated right. GGA functionals are supported, even for open-shell. SIC works for Linear Transit calculations. Frozen core SIC potentials are computed (GGA). The code is not well suitable for treating d and f electrons within the valence shell. Ds and Fs are fine within the frozen core. The SIC code does not allow parallel calculations (yet). Geometry optimizations are not yet possible. SIC in combination with spin-orbit coupling is not possible yet.

Usage of the block key SICOEP:

```
SICOEP
 {IPRINT n}
 {NOLOCALIZE}
 {LOCALIZE {thrs}}
 {LDA name {Xa}}
 {GGA name}
 {SELFCONSISTENT n}
 {POSTSCF}
 {SKIPCYCLES every {start}}
 {STABLE frac}
 {NHOMO Nalpha {Nbeta}}
 {CORE cor}
 {DENSITY mode}
 {SHIPV filename}
 {READV filename}
 {READMOS filename {FREEZE}}
 {WRITEMOS filename}
```

```
  {NAILCANONICALS {eps}}
  {NPFITS n}
 end
```

IPRINT n

-2 = no printing except for fatal errors
-1 = condensed printing
0 = normal printing
1 = verbose printing
5 = debug printing
10 = exorcism printing

NOLOCALIZE

Calculate SIC energies for canonical MOs. Except for atoms and certain small molecules, this is guaranteed to produce non-optimal SIC energies, and/or lead to severe convergence problems.

LOCALIZE {thrs}

Calculate SIC energies for localized MOs. In this version, Boys-Foster procedure is used to obtain localized orbitals. By default, all occupied orbitals will participate in the Boys-Foster procedure. This can be changed by supplying thrs parameter, which will only allow mixing of canonical orbitals within thrs eV of each other. Calculate SIC energies for canonical MOs. Except for atoms and certain small molecules, this is guaranteed.

LDA name {Xa}

Requests a specific local density functional in Perdew-Zunger energy correction and KLI contribution to the XC potential (If name is XALPHA, the default alpha of 0.7 can be changed by specifying the additional argument). Normally, SIC code will use LDA functional requested (explicitly or implicitly) in the XC keyblock. Specifying 'NONE' with suppress LDA contribution in SIC energy and potential. Because the Perdew-Zunger energy correction, and the corresponding term in the XC potential are trying to remove a non-physical self-interaction contribution from the Kohn-Sham E(XC)/v(XC) contribution, this key should only be used for debugging.

GGA name

Requests a specific gradient-corrected functional in the SIC part, overriding the default choice (same as in 'XC' keyblock). Specifying 'NONE' will suppress GGA contribution to the SIC energy and potential. Note that very little input checking is done for this key. It is possible to specify a non-existent functional here, which will lead to unpredictable results. Use this key only for debugging.

SELFCONSISTENT {n}

Includes KLI contribution in the XC potential. This is the default. In case of convergence difficulties, v(KLI) will be recomputed until n-th SCF cycle (90 cycles by default). All subsequent SCF cycles will use v(KLI) from the n-th cycle. If a SIC calculation runs into convergence difficulties, it is important to make sure that the SIC energy does not change significantly between the last cycle where potential was computed, and the final SCF cycle.

POSTSCF

Calculates Perdew-Zunger energy correction, using orbitals from standard Kohn-Sham calculation (possibly after localizing them - see LOCALIZE/NOLOCALIZE). Except in a few special cases, such as atoms or the hydrogen molecule, post-SCF SIC corrections are not reliable (see S. Patchkovskii and T. Ziegler, JCP 116, 7806, Ref.[48]).

SKIPCYCLES every {start}

Requests that v(KLI) is recomputed after a given number of SCF iterations. By default, v(KLI) is recomputed on each SCF iteration (every=1). Because v(KLI) evaluation is expensive, using 2 or 3 here may lead to a reduction in computation time. If an optional second parameter is included, v(KLI) is omitted during the first few SCF cycles. This may reduce calculation time if starting guess is not very accurate. Because SIC energy expression is defined as a functional of orbitals, v(KLI) is always omitted in the first SCF cycle, regardless of the "start" setting.

`STABLE frac`

Specifies mixing coefficient for v(KLI) contribution to exchange-correlation potential. On each iteration where v(KLI) is recomputed, the it is updated as: v(n) = frac*v(KLI) + (1-frac)*v(n-1). The default for frac is 1.0 (i.e. no mixing). Supplying a value smaller than 1 may improve SCF convergence.

`NHOMO Nalpha {Nbeta}`

Chooses treatment of the free parameter in the KLI approximation (see S. Patchkovskii, J. Autschbach, and T. Ziegler, JCP 115, 26, Ref.[47]). The default is to choose the free parameter such that per-orbital potential shifts are non-negative. If this key is supplied, the per-orbital shifts of Nalpha/Nbeta highest occupied orbitals will be set to zero instead. Both choices are usually identical for converged solutions, but the default exhibits much better convergence behavior.

`CORE cor`

cor can be one of:
`IGNORE`: Ignores contributions of frozen core orbitals to SIC energy and potential. Only valence orbitals will appear in KLI potential mixing expression (eq. 14 of [49]).
`RHO`: Includes core density in the total density of eq.14, but ignore core orbitals otherwise.
`POTENTIAL`: Includes SIC potential of the core orbitals in v(KLI), but sets corresponding potential shifts to zero.
`FULL`: Full treatment of the frozen core - frozen core orbitals participate in KLI potential equilibration (eq. 16) on the equal footing with the valence orbitals. This is the default.
For POTENTIAL and FULL, tesseral harmonics are used for the angular part of the core orbitals.

`DENSITY mode`

This keyword controls evaluation of per-orbital densities in a SIC calculation. mode can be one of:
`EXACT`: Evaluate densities from molecular orbitals. This is the default.
`FIT`: Evaluate densities from auxiliary fits. Using this option is not recommended - it is both slower, and less accurate than EXACT.

`SHIPV filename`

Write v(KLI) contribution to XC potential on grid, to an external file. When using this option, it is a good idea to write out the numerical integration grid as well, by adding "SAVE TAPE10" to the ADF input file.

`READV filename`

v(KLI) is taken from an external file, and add it to the XC potential. Current integration grid must match the grid used to calculate the potential. The only certain way to guarantee this is to save the integration grid on TAPE10, and pass it around, together with the v(KLI) potential. Specifying this keyword will deactivate add other processing in SIC code, including calculation of Perdew-Zunger energies and SIC potential updates.

`READMOS filename {FREEZE}`

For the first evaluation of the v(KLI) potential, loads localized orbitals from the specified file, instead of localizing canonical Kohn-Sham MOs. Unless FREEZE is specified, subsequent SCF cycles will use localized canonical MOs.

`WRITEMOS filename`

Stores localized orbitals to an external file. Reading these orbitals back with READMOS provides a rudimentary restart capability.

`NAILCANONICALS {eps}`

Stabilize orientation of degenerate canonical MOs prior to localization. The default is not to stabilize. eps is the degeneracy criterion in eV (0.001 by default). Supplying this key may improve convergence when high local symmetry is present in a molecule.

`NPFITS n`

Number of fit coefficient sets to be computed in a single pass. Large values will improve performance, but need more memory. The default of 15 is usually adequate.

General remarks

- The phrase non-local in the discussion of density functionals does not mean that non-local potentials are involved.
  The potentials are perfectly local, but when you go beyond LDA and include gradient corrections, the value of the density functional potential in a point r is evaluated not only from the local value of the charge density, but also from the gradient of the charge density.
- The Stoll formula is considered to be a *correlation* correction to the *Local* Density Approximation.
  It is conceptually not correct to use the Stoll correction *and* apply non-local gradient (GGA) corrections to the correlation.
  It is the user's responsibility, in general and also here, to avoid using options that are not solidly justified theoretically.
- It is questionable to apply gradient corrections to the *correlation*, while not doing so at the same time for the exchange.
  Therefore the program will check this and stop with an error message.
  This check can be overruled with the key ALLOW.
- The issue of the 'best' density functional is a subject of extensive and widespread research.
  It is generally recognized that applying gradient corrections to the simplest Local Density Approximation usually gives better results for comparison with experimental data, especially as regards bond energies and the spectra computed from one-electron energies.
- The incorporation of gradient corrections during the SCF significantly increases the computing effort.
  In this respect it makes no difference which specific GGA formula is applied.
  The Energy (PostSCF) feature is therefore an alternative worthwhile considering:
  it saves a lot of time and the effects of this approximation are often small as regards the SCF solution, so the non-self-consistent aspect hardly shows up in the computed bond energy.
  In Geometry Optimizations, however, the Post-SCF option implies that the energy gradients are computed from
  the LDA energy expression and hence the resulting optimized geometry corresponds to the LDA functional.
  In such a case, including the GGA term may make a substantial difference to the computed equilibrium geometry.

Relativistic effects

```
RELATIVISTIC {level} {formalism} {potential}
```

`Level`

May be None (this suppresses the key, and is equivalent to not using the key at all), Scalar (default: scalar relativistic effects), or SpinOrbit (using double group symmetry).

`Formalism`

Pauli (default) or ZORA (ZORA is recommended!)

`Potential`

SAPA (default) or Full. The Full option is obsolete. It is here mainly for historical reasons. The SAPA method is described in Ref.[50] for the BAND program. The same potential is used in the ADF program. One may think that the Full option gives extra accuracy. However, this is not the case, it only leads to extra CPU time and extra DISK space usage.

The key RELATIVISTIC instructs ADF to take relativistic effects into account. By default (omission of the key) this is suppressed. Recommendation use: Relativistic Scalar ZORA or Relativistic SpinOrbit ZORA.

Pauli

Specification of the Pauli formalism means that the first order relativistic corrections (the Pauli Hamiltonian) will be used [51-60]. In a *scalar* relativistic run ADF employs the single point group symmetry and only the so-called *scalar* relativistic corrections, Darwin and Mass-Velocity. The treatment is not strictly first-order, but is *quasi*-relativistic, in the sense that the first-order scalar relativistic Pauli Hamiltonian is diagonalized in the space of the non-relativistic solutions, i.e. in the non-relativistic basis set.

The quasi-relativistic approach improves results considerably over a first-order treatment. There are, however, theoretical deficiencies due to the singular behavior of the Pauli Hamiltonian at the nucleus. This would become manifest in a complete basis set but results are reasonable with the normally employed basis sets. However, this aspect implies that it is not recommended to apply this approach with an all-electron basis set for the heavy atoms, and for very heavy elements even a frozen core basis set often fails to give acceptable results. The problems with the quasi relativistic approach of the Pauli Hamiltonian are discussed for example in Ref.[61].

ZORA

The ZORA approach gives generally better results than the Pauli formalism. For all-electron calculations, and in fact also for calculations on very heavy elements (Actinides), the Pauli method is absolutely unreliable. Therefore, with its formal introduction in ADF1999, the ZORA method is the recommended approach for relativistic calculations with ADF.

ZORA refers to the Zero Order Regular Approximation [61-65]. This formalism requires special basis sets, primarily to include much steeper core-like functions; applying the ZORA method with other, not-adapted basis sets, gives unreliable results. The ZORA basis sets can be found in the ADF database, in subdirectories under the $ADFHOME/atomicdata/ZORA directory.

The ZORA formalism can also be used in Geometry Optimizations. However, there is a slight mismatch between the energy expression and the potential in the ZORA approach, which has the effect that the geometry where the gradients are zero does not exactly coincide with the point of lowest energy. The differences are small, but not negligible, order of magnitude: 0.001 angstrom.

Spin-Orbit coupling

The Spin-Orbit option uses double-group symmetry. The symmetry-adapted orbitals are labeled by the

quantum number *J* rather than *L* and any references in input to subspecies, such as a specification of occupation numbers, must refer to the double group labels.

Create runs must *not* use the Spin-Orbit formalism. The SFO analysis of Molecular Orbitals for a Spin-Orbit calculation is only implemented in the case of a scalar relativistic fragment file, which is the whole molecule. Gradient calculations for the Spin Orbit formalism have not yet been implemented. Therefore, computations of harmonic frequencies calculation and geometry optimizations cannot employ this feature.

In a Spin-Orbit run each level can allocate 2 electrons (times the dimension of the irreducible representation) as in a normal restricted calculation. However, contrary to the normal case these two electrons are not directly associated with spin-α and spin-β, but rather with the more general Kramer's symmetry. Using the unrestricted feature in order to assign different numbers of electrons to a and b spin respectively cannot be applied as such. However, one can use the unrestricted option in combination with the collinear or noncollinear approximation. In that case one should use symmetry NOSYM, and each level can allocate 1 electron.

Relativistic core potentials

In all relativistic calculations - scalar as well as spin-orbit - the relativistic atomic core densities and the relativistic atomic potentials *must* be made available to ADF on a file specified with the key COREPOTENTIALS. Starting from the ADF2006.01 release this is necessary only in the 'create' run of the atoms. In the molecular calculation this key is not required anymore. If supplied then the file must contain data for *all* atom types in the molecule, even for those atoms where relativistic aspects are expected to be negligible or that may not have a frozen core at all (such as Hydrogen). Excepted are any Ghost atoms (for instance for a BSSE calculation): these can not have any core potentials. This is tested by the program, internally, by looking at the nuclear charge and at the number of electrons belonging to an atom: if both numbers are zero, no (relativistic or other) core potential is allowed. Also the potential used in the ZORA kinetic energy operator in the SAPA (sum of neutral atomic potential approximation) method should be present on this file (which will be the case if the program DIRAC is used to generate this file).

Relativistic potentials can and should be generated with the auxiliary program dirac, see the utilities document, and the examples.

As of ADF2003, the recommended way to generate atomic fragments and relativistic potentials is by using the BASIS keyword.

Solvent effects: COSMO

You can study chemistry in solution, as contrasted to the gas phase, with the implementation in ADF [66] of the Conductor like Screening Model (COSMO) of solvation [67-69]. The energy derivatives can also be calculated, so geometry optimization, harmonic frequencies, et cetera are available within this model.

The COSMO model is a dielectric model in which the solute molecule is embedded in a molecule-shaped cavity surrounded by a dielectric medium with given dielectric constant e. Energy-related terms are computed for a conductor first, then scaled by the function

$$f(\epsilon) = (\epsilon-1)/(\epsilon+x) \quad (2.1.1)$$

The empirical scaling factor *x* is specified in the input data block for the solvation key. The block key SOLVATION turns the solvation calculation on. In most cases default values are available for the involved parameters.

```
SOLVATION
 {Surf Esurf {NOKEEP}}
```

```
 {Div {Ndiv=3} {Min=0.5} {OFAC=0.8}}
 {NOASS}
 {Solv {Name=solvent}{Eps=78.4}{Del=1.4}{Rad=1.4}{Emp=0.0}{Cav0=1.321}{Cav1=0.0067639}}
 {RADII
 name1=value1
 name2=value2
  ...
 subend }
 {Charged {Method=meth} {Conv=1e-6} {Omega=1.0} (Iter=300} {Corr} }
 {C-Mat How {SCF} tol=1e-10 }
 {DISC {SC=0.01} {LEG=4} {TOL=0.1} }
 {SCF {When} {How}
 {LPRT }
End
```

Presence of the SOLVATION key block triggers the solvent calculation and does not require additional data. With subkeys you can customize various aspects of the model, for instance to specify the type of solute. None of the subkeys is obligatory. Follows a description of the subkeys

`Surf`

Esurf must be Wsurf, Asurf, Esurf or Klamt. Four different cavity types are available.
Wsurf triggers the Van der Waals surface (VdW), which consists of the union of all atomic spheres.
Asurf gives the Solvent-Accessible-Surface (SAS). This is similar to VdW but consists of the path traced by the center of a spherical solvent molecule rolling about the VdW surface or, equivalently, a VdW surface created by atomic spheres to which the solvent radius has been added. These two surface types contain cusps at the intersection of spheres.
Esurf (the default) gives the Solvent-Excluding-Surface (SES), which consists of the path traced by the *surface* of a spherical solvent molecule rolling about the VdW surface. Primarily, this consists of the VdW surface but in the regions where the spheres would intersect, the concave part of the solvent sphere replaces the cusp.
These 3 surfaces are constructed with the GEPOL93 algorithm [70]
The SES surface is the default in ADF.
The fourth surface option is Klamt as described in [67]. It excludes the cusp regions also.

The optional parameter NOKEEP controls surface creation during calculation of frequencies by numerical differentiation. By default, the surface is constructed only once at the central geometry and is used for the rest of the calculation. If the NOKEEP is specified then ADF will construct a new surface at each displaced geometry. The NOKEEP option was the default in ADF2005 and earlier versions but it was found to cause problems. Since ADF2006 one needs to specify SURF NOKEEP to get the same behavior.
The actual construction of the surface involves a few technical parameters controlled with the subkey

`DIV`

Ndiv controls how fine the spheres that in fact describe the surface are partitioned in small surface triangles, each containing one point charge to represent the polarization of the cavity surface. Default Ndiv=3
Min specifies the size, in angstrom, of the smallest sphere that may be constructed by the SES surface. For VdW and SAS surfaces it has no meaning. Default Min=0.5
Ofac is a maximum allowed overlap of new created spheres, in the construction procedure. Default Ofac=0.8

`NOASS`

By default all new spheres that are created in the surface-construction are assigned to atoms, for the purpose of gradient computations (geometry optimization). Specifying the noass subkey turns this off. It has no argument.

`Solv`

Solvent details. Eps specifies the dielectric constant (the default relates to water). Rad specifies the radius of the (rigid sphere) solvent molecules, in angstrom. Instead of specifying Eps and Rad one can specify a solvent name or formula after 'name='. The following table lists names and formulas that are recognized with the corresponding values for Eps and Rad. The names and formulas are case-insensitive.

Name	Formula	Eps	Rad
AceticAcid	CH3COOH	6.19	2.83
Acetone	CH3COCH3	20.7	3.08
Acetonitrile	CH3CN	37.5	2.76
Ammonia	NH3	16.9	2.24
Aniline	C6H5NH2	6.8	3.31
Benzene	C6H6	2.3	3.28
BenzylAlcohol	C6H5CH2OH	13.1	3.45
Bromoform	CHBr3	4.3	3.26
Butanol	C4H9OH	17.5	3.31
isoButanol	(CH3)2CHCH2OH	17.9	3.33
tertButanol	(CH3)3COH	12.4	3.35
CarbonDisulfide	CS2	2.6	2.88
CarbonTetrachloride	CCl4	2.2	3.37
Chloroform	CHCl3	4.8	3.17
Cyclohexane	C6H12	2	3.5
Cyclohexanone	C6H10O	15	3.46
Dichlorobenzene	C6H4Cl2	9.8	3.54
DiethylEther	(CH3CH2)2O	4.34	3.46
Dioxane	C4H8O2	2.2	3.24
DMFA	(CH3)2NCHO	37	3.13
DMSO	(CH3)2SO	46.7	3.04
Ethanol	CH3CH2OH	24.55	2.85
EthylAcetate	CH3COOCH2CH3	6.02	3.39
Dichloroethane	ClCH2CH2Cl	10.66	3.15
EthyleneGlycol	HOCH2CH2OH	37.7	2.81
Formamide	HCONH2	109.5	2.51
FormicAcid	HCOOH	58.5	2.47
Glycerol	C3H8O3	42.5	3.07
HexamethylPhosphoramide	C6H18N3OP	43.3	4.1
Hexane	C6H14	1.88	3.74
Hydrazine	N2H4	51.7	2.33
Methanol	CH3OH	32.6	2.53
MethylEthylKetone	CH3CH2COCH3	18.5	3.3
Dichloromethane	CH2Cl2	8.9	2.94
Methylformamide	HCONHCH3	182.4	2.86
Methypyrrolidinone	C5H9NO	33	3.36
Nitrobenzene	C6H5NO2	34.8	3.44
Nitrogen	N2	1.45	2.36

Nitromethane	CH3NO2	35.87	2.77
PhosphorylChloride	POCl3	13.9	3.33
IsoPropanol	(CH3)2CHOH	19.9	3.12
Pyridine	C5H5N	12.4	3.18
Sulfolane	C4H8SO2	43.3	3.35
Tetrahydrofuran	C4H8O	7.58	3.18
Toluene	C6H5CH3	2.38	3.48
Triethylamine	(CH3CH2)3N	2.44	3.81
TrifluoroaceticAcid	CF3COOH	42.1	3.12
Water	H2O	78.39	1.93

Emp addresses the empirical scaling factor *x* in the formula above.
Other options specify a linear parameterization of non-electrostatic terms as a function of surface area:

$E_{\text{non-elst}} = f(\epsilon) * (\text{CAV0} + \text{CAV1} * \textit{area})$ (2.1.2)

In order to construct the surface you have to specify the atomic ('Van der Waals') radii. There are three ways of doing this. In the first method you append 'R=value' to the atomic coordinates record, in the ATOMS key block. This would look like, for instance

```
C 1 2 3 CC CCO CCOH f=C.dz R=2.0
```

It assigns a radius of 2.0 to the Carbon atom.
In the second method you apply the same format, but specify a symbol (identifier) rather than a value

```
C 1 2 3 CC CCO CCOH f=C.dz R=C-sp3
```

The identifiers must be defined in the (optional) RADII subkey block in the Solvation data block (see next).
In the third method, you don't modify the Atoms block at all. In this case, the RADII subkey must be used and the 'identifiers' in it must be exactly the atom type names in the Atoms block.

`Radii`

This subkey is block type. Its data block (if the subkey is used) must terminate with a record subend. In the Radii data block you give a list of identifiers and values

```
SOLVATION
  ...
 Radii
 name1=value1
 name2=value2
  ...
 Subend
  ...
End
```

The values are the radii of the atomic spheres, in the same units of length as used in the Atoms block (angstrom or bohr). The names specify to which atoms these values apply. As discussed for the Solv subkey this depends on the Atoms block. If in the specification of atomic coordinates you have used the 'R=' construct to assign radii, with identifiers rather than values for the R-value, these identifiers must be defined in the Radii sub block. If no 'R=' construct was applied in the Atoms block, you must use the atom type names as they occurred in the Atoms data block. Referring to the example given in the Solv subkey discussion, you might have

```
...
 Radii
  C-sp3=2.0
  ...
 Subend
...
```

A simple atom type reference might look like

```
...
 Radii
  C=2.0
  ...
 Subend
...
```

When no radius specified a default value is used. The default value for an atom is the corresponding Van der Waals radius from the MM3 method by Allinger divided by 1.2.

This concludes the discussion of the Radii subkey.

CHARGED

This addresses the determination of the (point) charges that model the cavity surface polarization. In COSMO calculations you compute the surface point charges *q* by solving the equation A*q*=-*f*, where *f* is the molecular potential at the location of the surface charges *q* and A is the self-interaction matrix of the charges. The number of charges can be substantial and the matrix A hence very large. A direct method, i.e. inversion of A, may be very cumbersome or even impossible due to memory limitations, in which case you have to resort to an iterative method.
Meth specifies the equation-solving algorithm. Meth=INVER requests direct inversion. Meth=GAUS calls for the Gauss-Seidel iterative method. Meth=Jacobi activates another standard iterative procedure. The latter two methods require a positive-definite matrix (which may fail to be the case in an actual calculation) and can be used with a relaxation technique, controlled by the relaxation parameter OMEGA (1.0=no relaxation). Meth=CONJ (default) uses the preconditioned biconjugate gradient method. This is guaranteed to converge and does not require huge amounts of memory.
CONV and ITER are the convergence criterion and the maximum number of iterations for the iterative methods.

Some of the molecular electronic charge distribution may be located outside the cavity. This affects the assumptions underlying the COSMO equations. Specifying the CORR option to the CHARGED subkey constrains the computed solvent surface charges to add up to the negative of the molecular charge.

C-MATRIX

- How: For the potential *f* we need the Coulomb interaction between the charges *q* and the molecular electronic density (and nuclei). Three methods are available, specified by the first option to the C-Matrix subkey.
a) EXACT: compute the straightforward Coulomb potential due to the charge *q* in each point of the molecular numerical integration grid and integrate against the electronic charge density. This is, in principle, exact but may have inaccuracies when the numerical integration points are very close to the positions of a charge *q*. To remedy this, the point charges *q* can be 'smeared out' and represented by a disc, see the next subkey DISC.
b) FIT: same as EXACT, but the *q*-potentials are now integrated not against the exact electronic charge density, but against the (much cheaper-to-compute) fitted density. The same DISC considerations apply.
c) POT: evaluate the molecular potential at the position of the charge *q* and multiply against the *q*-strength. Since the molecular Coulomb potential is computed from the fit density, any difference in results between the FIT and the POT approach should be attributed to the DISC issue.

POT is the default, because it is faster, and is only inadequate if the fit density is very inaccurate, which would be a problem anyway.
- SCF: If you specify this option, the computation of the Coulomb interaction matrix (between electrons and surface charges) is carried out during the SCF procedure, but this turns out to hamper the SCF convergence behavior. Therefore: not recommended. *IF* you use it, the program will switch to one of the other 3 methods, as given by the 'How' option, as soon as the SCF convergence error drops below
TOL: (applies only to the SCF option, which is not recommended).

`DISC`

Applies only when the C-matrix method is EXACT or FIT. Note, however, that the default for the C-matrix method is POT, in which case the DISC subkey has no meaning. The DISC key lets the program replace the point charges *q* by a solid uniformly charged spherical surface disc whenever the numerical integration accuracy requires so, i.e. for those charges that are close to numerical integration points.
Options:
SC defines a shrinking factor, by which the actual disc radius used is reduced from its 'normal' value: an inscribed disc in the triangular surface partitions that define the distribution of surface charges, see the subkey DIV.
LEG gives the polynomial expansion order of the disc potentials. The Legendre expansion converges rapidly and the default should be adequate.

TOL is a tolerance parameter to control the accuracy of the disc potential evaluations.

`SCF`

In COSMO calculations you can include the surface charges in the Fock operator self-consistently, i.e. by recomputing the charges *q* at every SCF cycle and include them in the equations, or in a perturbational manner, i.e. post-SCF. This is controlled with the first option. The When option must be either VAR or PERT, for variational and perturbational, respectively. Default is VAR.
The second (HOW) option applies only to the WHEN=VAR case and may affect the speed of SCF convergence. The COSMO calculation implies a considerable increase in CPU time! Values for HOW:
- ALL: This includes it in all SCF cycles (except for the first SCF cycle, which is gas-phase)
- LAST: This lets the program first converge the SCF completely without any solvent effects. Thereafter, the COSMO is turned on, hopefully converging in fewer cycles now, to compensate for the 'double' SCF effort.
- TOL=0.1 (or another value) is an in-between approach: converge the gas-phase SCF until the SCF error is below TOL, then turn on COSMO.

`LPRT`

This is a debug switch and triggers a lot more output related to the cavity construction etc.

Discrete Solvent Reaction Field Model: DRF

The Discrete Solvent Reaction Field (DRF) model is a hybrid Quantum mechanical and Molecular Mechanics (QM/MM) model for studying solvation effects on (time-dependent) molecular properties such as dipole moments, excitation energies and (hyper)polarizabilities[141-145]. The classical solvent molecules are represented using distributed atomic charges and polarizabilities.

DRF Theory

Within the Discrete Solvent Reaction Field model the QM/MM operator is

$$H_{QM/MM} = \sum_i v^{DRF}(r_i,\omega) = \sum_i [v^{el}(r_i) + v^{pol}(r_i,\omega)$$

where the first term, v^{el}, is the electrostatic operator and describes the Coulombic interaction between the QM system and the permanent charge distribution of the solvent molecules. The second term, v^{pol}, is the polarization operator and describes the many-body polarization of the solvent molecules, i.e. the change in the charge distribution of the solvent molecules due to interaction with the QM part and other solvent molecules. The charge distribution of the solvent is represented by atomic point charges and the many-body polarization by induced atomic dipoles at the solvent molecules. The induced atomic dipole at site *s* is found by solving a set of linear equations

$$\mu^{ind}_{s,\alpha}(\omega) = \alpha_{s,\alpha\beta} [F^{init}_{s,\beta}(\omega) + \sum_{t\neq s} T^{(2)}_{st,\beta\gamma} \mu^{ind}_{t,\gamma}(\omega)],$$

where $\alpha_{s,\alpha\beta}$ is a component of the atomic polarizability tensor at site *s* The screened dipole interaction tensor is given by

$$T^{(2)}_{st,\beta\gamma} = 3f^{T}_{st}R_{st,\alpha}R_{st,\beta}/R^{5}_{st} - f^{E}_{st}\delta_{\alpha\beta}/R^{3}_{st}$$

where the damping functions f^{T}_{st} and f^{E}_{st} have been introduced, see also [146]. A smeared-out point charge model [147] is used for short-range damping of the QM/MM operator

$$1/R_{st} \rightarrow 1/S_{st} = erf(R_{st})/R_{st}$$

The scaled distance, S_{st}, then replaces the normal distance, R_{st}, in the QM/MM operator.

Parameters needed in the DRF model

In order to perform a DRF calculation two types of parameters (model atomic charges and atomic polarizabilities) for each type of atom in the MM part are required. The point charges should represent at least the permanent molecular dipole moment, and the distributed atomic polarizabilities the full molecular polarizability tensor. The atomic charges can straightforward be obtained using e.g. Multipole Derived Charges (MDC) [See section MDC] and the distributed polarizabilities by adopting standard parameters or refitting them to match the calculated polarizability tensor [146,147]. This allows for a simple procedure to obtain the solvent model parameters which subsequently can be used in the DRF calculation.

DRF input

To run a DRF calculation two block keys, `DRF` and `EXTERNALS`, are required. The `DRF` key controls the general options and the `EXTERNALS` key provides the atomic data for the MM atoms. The `DRF` key **should not** be used in combination with: COSMO, QM/MM, geometry optimization, frequency calculation or point charges.

A DRF calculation are invoked by the following block key `DRF`:

```
DRF
 {NORESP}
 {NOPOL}
 {NOCHAR}
 {LOCALFIELD}
 {EFIELD ex ey ez &}
 {AFAC a}
end
```

`NORESP`

DRF is ignored in RESPONSE and EXCITATION

NOPOL

The many-body operator is ignored (no induced dipoles)

NOCHAR

The Coulomb operator is ignored (no charges)

LOCALFIELD

Local Fields are included

EFIELD

Homogeneous external electric field is included. ex value of field in x-direction, ey value of field in y-direction, and ez value of field in z-direction. A continuation symbol, &, is required. The units are atomic units.

EXTERNALS

The EXTERNALS block key controls the input data for the MM atoms. If this block is not found no DRF calculation will be performed. For each MM atom the following data are required:

```
EXTERNALS
 atm num grp-nam grp-num, char, x, y, x, pol
 ...
 GROUP
 {...}
end
```

atm

Type of atom, i.e., H, O, ...

num

number of atoms (optional)

grp-nam

Name of the group to which the atom belongs

grp-num

Number of the group to which the atom belong

char

atomic charge (in atomic units)

x

x-coordinate

y

y-coordinate

z

z-coordinate

pol

atomic polarizability (in atomic units)

GROUP

Indicates the end of group

The separation of molecules into GROUP's are important. Since in the many-body polarization operator only inter-molecular interactions, i.e. only interaction between sites which do not belong to the some group, are included. Therefore, it is important that the combined string (grp-nam + grp-num) is unique for each GROUP.

An example of a EXTERNALS block for two water molecules:

```
EXTERNALS
 O  4 water    2, -0.6690, -11.380487, -11.810553,  -4.515226,  9.3005
 H  5 water    2,  0.3345, -13.104751, -11.837669,  -3.969549,  0.0690
 H  6 water    2,  0.3345, -10.510898, -12.853311,  -3.320199,  0.0690
 GROUP
 O  7 water    3, -0.6690,  -1.116350,   9.119186,  -3.230948,  9.3005
 H  8 water    3,  0.3345,  -2.822714,   9.717033,  -3.180632,  0.0690
 H  9 water    3,  0.3345,  -0.123788,  10.538199,  -2.708607,  0.0690
 GROUP
 {...}
end
```

Frozen-Density Embedding: FDE

In the orbital-free frozen-density embedding (FDE) formalism [184], the environment of an embedded subsystems is accounted for by means of the embedding potential depending explicitly on electron densities corresponding to the embedded subsystem (e.g., a solvated molecule) and its environment (e.g., solvent). For a detailed review, see Ref. [205]. The ADF implementation of the method is described in detail in Ref. [185].

A time-dependent linear-response generalization of this embedding scheme was derived in Ref. [186]. Its implementation in an approximate form, which assumes a localized response of the embedded system only, is described in the supplementary material to Ref. [187]. For possible drawbacks and pitfalls in connection with this approximation, see Refs. [185,190,193].

The current implementation in ADF allows the calculation of molecular properties that only depend on the electron density and of response properties using TDDFT. For an application to the calculation of several molecular properties in solution and a comparison to the DRF model also available in ADF, see Ref. [190]. For further applications of the ADF implementation, see Ref. [189] (weakly interacting complexes) and Refs. [185,190-192, 206-207] (solvent effects).

FDE Input

A frozen density calculation can be invoked with the block key FDE. In the simplest case, this input key should look like this:

```
FDE
```

```
  TAPE21FD filename
  PW91K
end
```

This will invoke a frozen-density embedding calculation, in which the frozen density is imported from the TAPE21 file (filename). Furthermore, the recommended PW91k functional is chosen for the nonadditive kinetic energy. For all other options the defaults will be used.

In the current implementation, only the electron density of the embedded system is calculated. Therefore, only properties that depend directly on the electron density (e.g., dipole moments) are available. In particular, the calculation of interaction energies or of energy gradients is not implemented yet.

The TDDFT extension of the FDE formalism allows the calculation of electronic excitation energies and polarizabilities. This extension is automatically activated if FDE is used in combination with the EXCITATIONS or the RESPONSE key.

To prevent possible problems, FDE calculations should use no symmetry (SYMMETRY NOSYM). However, it is possible to use symmetry in FDE calculations by adding the keyword ALLOW FDESYM. In this case, the user has to take care that the same symmetry that is used in the FDE calculation was used in the preparation of the frozen density. In addition, the same standard orientation has to be used in both calculations. None of this is checked by the program.

There a a number of options available in FDE calculations that will be described in the following.

```
FDE
  TAPE21FD filename
  {DENSTYPE [SCF | FullSum]}
  {kinetic energy functional}
  {GGAPOTXFD exchange functional}
  {GGAPOTCFD correlation functional}
  {FULLGRID}
end
```

`TAPE21FD filename`

The filename of the TAPE21 file of the calculation of the frozen density is given with the TAPE21FD option. The nuclear coordinates and the electron density will then be imported from this file. By default, the converged SCF electron density will be imported from the TAPE21FD file. Different ways of preparing the frozen density are described in the following section.

`DENSTYPE`

The DENSTYPE option can be used to specify which electron density will be imported from the TAPE21FD file as the frozen density. The following options are available:

`DENSTYPE SCF (default)`

The final SCF density is used.

`DENSTYPE FullSum`

The superposition of the densities of the initial fragments is used. This is in particular useful in combination with the DENSPREP option described in the next section. (If the TAPE21FD file was produced in a complete SCF calculation, the results of this calculation is ignored)

`kinetic energy functional`

There are several approximate kinetic energy functionals available, that can be used for the nonadditive kinetic energy in the effective embedding potential. If no kinetic energy functional is specified, by default the

local-density approximation (Thomas-Fermi functional) is used. For an assessment of functionals for weakly interacting systems see Ref. [188]. Based on this study, the use of PW91k is recommended.

`THOMASFERMI (default)`

Thomas-Fermi LDA functional [194,195]

`WEIZ`

gradient-dependent von Weizsäcker functional [196] (no LDA part included)

`TF9W`

Thomas-Fermi functional + 1/9 von Weizsäcker functional

`PW91k (recommended functional)`

GGA functional based on PW91 exchange functional, reparametrized for the kinetic energy by Lembarki and Chermette [197]

`LLP91`

GGA functional based on Becke88 exchange functional, reparametrized for the kinetic energy by Lee, Lee, and Parr [198]

`PW86k`

GGA functional based on PW86 exchange functional [199]

`OL91A, OL91B`

gradient-dependent functionals OL1 and OL2 by Ou-Yang and Levy [200]

`THAKKAR92`

gradient-dependent functional by Thakkar [201]

`COULOMB`

This option does not stand for a kinetic energy functional, but it disables the nonadditive kinetic energy part and the exchange-correlation part in the embedding potential. The remaining embedding potential will only contain the Coulomb interaction with the frozen density. Note that the use of this option is not recommended, it is useful for analysis purposes only.

`GGAPOTXFD`
`GGAPOTCFD`

By default, in the construction of the effective embedding potential the exchange-correlation functional that was specified in the XC block is used. It is possible to specify a different functional with the GGAPOTXFD and GGAPOTCFD options. This is particularly useful in combination with the use of model potentials like SAOP, that can not be used in the embedding potential because of their orbital dependence. (For a discussion, see Ref. [189])

`GGAPOTXFD exchange functional`

The exchange functional is used in the construction of the embedding potential. The same exchange functionals as in the XC key are available.

`GGAPOTCFD correlation functional`

The correlation functional is used in the construction of the embedding potential. The same correlation

functionals as in the XC key are available.

FULLGRID

By default, FULLGRID is not used, and in FDE calculations the integration grid is generated as described in Ref. [185] by including only atoms of the frozen system that are close to the nonfrozen system in the generation of the integration grid. The distance cutoff used is chosen automatically based on the extend of the basis functions of the nonfrozen system. (It can also be chosen manually, see the option qpnear in the INTEGRATION key) This scheme results in a efficient and accurate integration grid. However, it is possible that the default integration scheme is not accurate enough. This can be the case for weakly interacting systems and when the distance between the frozen and the nonfrozen system is large. It is therefore recommended to check the quality of the default integration grid by comparing to results obtained using the full supermolecular grid (FULLGRID option).

If the subkey FULLGRID is included all atoms of the frozen system are included in the generation of the integration grid. This results in the same grid that would be used in a supermolecular calculation of the combined frozen and nonfrozen system. The integration grid generated by this option might be much larger than the default grid. This option should be used to check the quality of the default integration grid.

Preparation of frozen density

The default way to construct a frozen density is to run a KS-DFT calculation with ADF for the embedding (outer) system, save the TAPE21 file, and use it with DENSTYPE SCF in a FDE calculation on the embedded system.

In Refs. [191], it was shown that for particular systems also approximate frozen densities can be used without significant loss of accuracy in the desired property. These approximations can include:

- a) the use of a simpler XC-functional in the KS-DFT calculation to construct the frozen density
- b) less strict SCF convergence criteria in this preparation step
- c) a superposition of molecular fragments
  (if the embedding system can be decomposed into weakly interacting units).

While methods a) and b) can be readily applied in normal KS-DFT calculations, method c) requires an efficient calculation of the frozen density as a superposition of molecular fragment densities, which can be done using ADF as follows:

- 1) The SCF densities for each of the individual molecular fragments (e.g., solvent molecules) have to be calculated, and their TAPE21 files need to be saved.
  This step is rather quick, especially if the solvent molecules are small.
- 2) An ADF input has to be prepared for the full solvent system, in which not atomic, but the molecular fragment files created in step 1) are used as initial fragments.
- 3) The keyword DENSPREP has to be specified in this input file.

```
DENSPREP
```

When this DENSPREP-calculation is executed, ADF will stop directly after setting up a sum-of-fragment density for the molecular fragments. A rudimentary TAPE21-file is written, which contains the density information in a special section. This density information can be used in a subsequent FDE calculation by using the 'DENSTYPE FULLSUM' option.

DENSPREP calculations can be combined with partial freeze-and-thaw relaxation steps if fragments are used which do not consist of isolated solvent molecules, but of (one or several) solvent molecule fragments which have been created in FDE calculations themselves. For an example of this technique, see Ref. [185].

Freeze-and-thaw cycles

The FDE scheme can be applied in so-called "freeze-and-thaw cycles", in which the electron density of both subsystems is optimized by interchanging the role of the frozen and nonfrozen density until convergence is reached. This can be used to improve an approximate frozen density or fully self-consistent as an alternative to conventional KS-DFT.

With the current implementation, the FDE scheme can be applied in freeze-and-thaw cycles by running successive (separate) FDE calculation in which the density obtained from the previous calculation (on the other subsystem) is used as the frozen density.

In the Examples document there is an example which shows how to setup fully self-consistent FDE calculations using freeze-and-thaw cycles.

Restrictions and pitfalls

In the current implementation, only the electron density of the embedded system is calculated. Therefore, only properties that depend directly on the electron density (e.g., dipole moments) are available. In addition, the TDDFT extension allows the calculation of electronic excitation energies and polarizabilities.

EVERYTHING ELSE IS NOT YET IMPLEMENTED. THE RESULTS OBTAINED FOR OTHER PROPERTIES MIGHT BE MEANINGLESS.

In particular, interaction energies and energy gradients are not yet available.

Kinetic energy functional:
Although the effective embedding potential is derived from first principles using universal density functionals, the ADF implementation relies on approximations. Currently, two implemented approximations are recommended [188]: PW91K, which uses electron densities and the corresponding gradients to express the non-additive kinetic energy component of the embedding potential, or TF (Thomas-Fermi LDA functional), which does not use gradients at all. Either approximation is applicable only in cases where the overlap between electron densities of the corresponding interactions is small. Note: so far, no approximation has been developed for the strong-overlap case - two subsystem linked by covalent bonds for instance.

Use of Symmetry:
In the case that symmetry is used in FDE calculations, the same symmetry and standard orientation as in the preparation of the frozen density must be used. See the explanation above.

Basis set in FDE calculations:
In FDE calculations, only the basis functions of the nonfrozen system are used, no basis functions of the frozen system are included. This type of basis set expansion for FDE calculations is usually labeled "monomer expansion". Currently it is not possible to include basis functions located on the frozen system.

The FDE implementation in ADF is work in progress. The input format and keywords described here will change in future versions of ADF

Electric Field: Homogeneous and Point Charges

A homogeneous external electric field and/or the field due to point charges can be included in the Fock operator. Either can be applied only in a Single-Point calculation (or a Create run) because the energy derivatives that are computed in Geometry Optimizations do not take the fields into account.

```
EFIELD {ex ey ez}
 {x y z q
```

```
 x y z q
  ...
end }
```

EFIELD

This *general* key can be used as a simple key or as a block key. The block form applies if no argument (ex, ey, ez) is given or when the argument is followed by the *continuation symbol* (&).

ex,ey,ez

Define a homogeneous electric field in atomic units: atomic Volts per bohr; the relation to SI units is: 1a.u. = 5.14 ... *10{V/m}.
The units applied by adf for the interpretation of homogeneous field values are not affected by any units used for specifying atomic coordinates. By default no homogeneous E-field is included.

x, y, z, q

The Cartesian coordinates and strength of a point charge (in elementary charge units, +1 for a proton). Each point charge must be specified on a separate line in the data block. The Cartesian coordinates are in the units of length that was set by units (for interpreting atomic coordinates input). By default no point charges are included.

Orientation of the fields

When the atomic coordinates are input in z-matrix format, the direction of the homogeneous field and the location of the point charges as specified in input are interpreted as referring to the standard Cartesian frame associated with z-matrix input. The standard frame means: the first atom at the origin, the second on the positive x-axis, the third in the xy-plane with positive y.

If the program rotates (and translates, as the case might be) the atoms from the input frame - or the auto-generated frame in case of z-matrix input - to some other frame, for instance to accommodate the internal ADF symmetry orientation requirements, the fields are transformed along with the atoms.

Symmetry

The homogeneous electric field and the point charge fields may polarize the electronic charge density. This must be accounted for in the point group symmetry. If symmetry is not specified in input, the program computes the symmetry from the nuclear frame and the fields.

Bonding energy

The bonding energy is computed as: the energy of the molecule in the field minus the energy of the constituent fragments in the same field. Of course, the fragments may not be polarized and hence not be self-consistent in this field. This depends on how the fragments themselves were computed.

Polarizability and hyperpolarizability

ADF supports a direct calculation of the (hyper) polarizability (see next section). The static (hyper) polarizabilities can also be computed by applying a small homogeneous field and comparing the results with the field-free data.

MM Dispersion

The idea to get an accurate description of van der Waals complexes by density functional theory by including empirical corrections by Grimme [211] was implemented in ADF by J.M. Ducere from the group of Prof. L. Cavallo [215]. Please contact this group for more details on this functionality.

This is an expert option. As input one needs certain atomic parameters and (for a given basis set and functional optimized) parameters for a damping function. At the moment only for a few atoms atomic parameters can be found in the file $ADFRESOURCES/MMDispersion/disp-param. Only for the PBE functional with a DZP or TZP basis set parameters are optimized for the damping function. This optimization was done with respect to MP2 theoretical data. The parameters from Grimme's paper can also be used.

```
MMDispersion
  {FILE_NAME  filename}
  {DAMPING    damping}
  {DAMP_PARAM damp_param {a b c}}
  {COMBI      combi}
  {DISPALL}
  {NODEFAULT}
  {ATOMTYPE
    attype  c6  pol  rad
  SUBEND}
End
```

`FILE_NAME filename`

Optional. The filename (full path) from which are read the C6 parameters, polarizabilities and radii. The file is expected to have the following structure:

```
 attype  c6  pol  rad
```

The attype must exactly match the atom-type name present in the ATOMS key-block (case-sensitive), for being recognized; c6, pol, and rad are in atomic units (hartree and bohr). A "---" sequence indicates the end of the read part. Even if the sqrt option is chosen for COMBI, a polarizability is needed. If the environment variable ADFRESOURCES is set, the default value for filename is $ADFRESOURCES/MMDispersion/disp-param.

`DAMPING damping`

Optional. Defines the kind of damping function to be used, damping can be one of:
sigm: sigmoid (default)
fermi: Fermi-like function (Grimme [211])

`DAMP_PARAM damp_param {a b c}`

Optional. Defines which parameters of the damping function should be used, damp_param can be one of:
tz: parameters optimized for PBE/TZP (default)
dz: parameters optimized for PBE/DZP
grimme: parameters from Grimme paper
cust a b c: parameters are a, b and c

`COMBI combi`

Optional. Defines the kind of combination rule to be used, combi can be one of:
s-k: Slater-Kirkwood combination rule (default)[214]
sqrt: square-root combination rule

DISPALL

Optional. If present, all atom-pairs are considered, else, only contributions from different fragments (different indexes, see below) are considered. DISPALL is NOT the default.

NODEFAULT

Optional. By default, if there is no match for a given atom-type, ADF looks in the parameter file specified in FILENAME for atomic default parameters (Grimme's ones). NODEFAULT switches off this check. Example: suppose the atom-type is H.text. By default if there is no match for H.text, but there is a match for H, parameters for H will be used. If NODEFAULT is set, and there is no match for H.text an error message is printed and ADF will stop.

ATOMTYPE

Optional. For input supplied c_6, polarizability and radius paramaters of atom-types; attype must exactly match an atom-type name present in the ATOMS block for being recognized; c_6, pol and rad are in a.u.

Atom-types and fragment-indexes are specified in the ATOMS keyblock:

```
ATOMS
  atom-type  x  y  z  FD=n
  ...
END
```

FD is the index of the fragment. FD=0 switch off the calculation for the atom. If DISPALL is present in the input, non-zero values of FD only have an analytical role. If DISPALL is not present, the contributions are calculated between atoms of different non-zero values of FD. By default, FD=1 for all atoms.

Time-dependent DFT: Excitation Energies, (Hyper) Polarizabilities

Excitation energies, frequency-dependent (hyper) polarizabilities, Van der Waals dispersion coefficients, higher multipole polarizabilities, Raman scattering intensities and depolarization ratios of closed-shell molecules are all available in ADF [71,72] as applications of time-dependent DFT (TDDFT) ; see [73] for a review.

New in ADF2004.01 is the calculation of circular dichroism (CD) spectra, and the calculation of the optical rotation (dispersion).

Starting from the ADF2005.01 version it is possible to calculate excitation energies for open-shell systems with TDDFT, including spin-flip excitation energies. New in ADF2005.01 is the possibility to use time-dependent current-density functional theory (TDCDFT).

New in ADF2006.01 is the possibility to calculate excitation energies for closed-shell molecules including spin-orbit coupling.

The input description for these properties is split in three parts: (a) general advice and remarks, (b) excitation energies, and (c) frequency-dependent (hyper) polarizabilities and related properties.

General remarks on the use of the TDDFT Response and Excitation functionality

Symmetry

As in calculations without TDDFT the symmetry is automatically detected from the input atomic coordinates

and need not be specified, except in the following case: infinite symmetries cannot be handled in the current release (ATOM, C(lin), D(lin)). For such symmetries a subgroup with finite symmetry must be specified in the input. The usual orientation requirements apply.
If higher multipole polarizabilities are required, it may also be necessary to use a lower subgroup (the program will stop with an error message otherwise). For verification of results one can always compare to a NOSYM calculation.

`Closed-shell`

The current implementation often supports only closed-shell molecules. If occupation numbers other than 0 or 2 are used the program will detect this, (but only at a later stage of the calculation) and abort. All 'RESPONSE' calculations must be spin-restricted.

`Open-shell`

Excitation energies can be obtained for open-shell systems in a spin-unrestricted TDDFT calculation. Spin-flip excitation energies can only be obtained in a spin-unrestricted TDDFT calculation.

`Atomic coordinates in a RAMAN calculation`

Atomic coordinate displacements in a RAMAN calculation must be Cartesian, not Z-matrix. Furthermore, the current implementation does not yet support constrained displacements, i.e. you must use *all* atomic coordinate displacements.

`Use of diffuse functions`

The properties described here may require diffuse functions to be added to the basis (and fit) sets. Poor results will be obtained if the user is unaware of this. As a general rule, diffuse functions are more important for smaller than for larger molecules, more important for hyperpolarizabilities than for normal polarizabilities, more important for high-lying excitation energies (Rydberg states) than for low-lying excitations, more important for higher multipole polarizabilities than for dipole polarizabilities. The user should know when diffuse functions are required and when they are not: the program will not check anything in this respect. For example, in a study on low-lying excitation energies of a large molecule, diffuse functions will usually have little effect, whereas a hyperpolarizability calculation on a small molecule is pointless unless diffuse functions are included. Diffuse even tempered basis sets are included in the ET/ directory of the database), for the elements H-Kr. Somewhat older basis sets can be found in the Special/Vdiff directory in the database. For some atoms diffuse basis sets may be available at the web site http:tc.chem.vu.nl/~vgisberg. For other atoms, the user will have to add diffuse basis and fit functions to the existing data base sets. It is not necessary to start from basis V as was done for the basis sets in Special/Vdiff. For example, for heavier elements it may be a good idea to start from the ZORA/QZ4P basis sets. It may be expected that even more extensive basis sets will come available in the future, when usage and experience increase.

`Linear dependency in basis`

If large diffuse basis sets are used, or if diffuse functions are used for atoms that are not far apart the calculation may suffer from numerical problems because of (near-) linear dependencies in the basis set. The user should be aware of this danger and use the DEPENDENCY key to check and solve this.

`The LINEARSCALING input keyword`

For reasons of numerical robustness and safety rather strict defaults apply for the neglect of tails of basis and fit functions (see the key LINEARSCALING) in a Response or Excitation calculation. This may result in longer CPU times than needed for non-TDDFT runs, in particular for larger molecules. Possibly this precaution is not necessary, but we have not yet tested this sufficiently to relax the tightened defaults.

`Relativistic effects`

The Response and Excitations options can be combined with scalar relativistic options (ZORA or Pauli). The

one-electron relativistic orbitals and orbital energies are then used as input for the property calculation. Spin-orbit effects have been incorporated only in this part of the code (excitation energies). In case of a ZORA calculation, the so-called 'scaled' orbital energies are used as default.

`Choice of XC potential`

For properties that depend strongly on the outer region of the molecule (high-lying excitation energies, (hyper) polarizabilities), it may be important to use a XC potential with correct asymptotic behavior (approaching -1/r as r tends to infinity).

Finally, several asymptotically correct XC potentials have In ADF the LB94 potential has been implemented for this purpose [15]. An alternative is the statistical average of orbital potentials (SAOP) [17].

With this particular XC functional, the XC potential is computed from the exact charge density for reasons of stability and robustness (whereas for other functions the (cheaper) fit density is used). This implies that computation times may be longer. Another 'side effect' is that, since there is no energy expression corresponding to the LB94 *potential*, the final (bonding) energy of a LB94 calculation uses another GGA and hence the energy result is not (exactly) consistent with the SCF procedure. Note, finally, that the LB94 potential is not suitable for geometry optimizations because it is rather inaccurate in the bonding region, see the discussion of the XC input key.
Applications with the LB94 potential to response calculations with ADF can be found in [74] (polarizabilities), [75-77] (hyperpolarizabilities), [78] (high-lying excitation energies), [79] (multipole polarizabilities and dispersion coefficients)

`Accuracy check list`

As mentioned before, the TDDFT module is relatively new and not extensively tested for a wide range of applications. Therefore, we strongly recommend the user to build experience about aspects that may affect the accuracy of TDDFT results. In particular we advise to 'experiment' with

- Varying integration accuracy

- Varying the SCF convergence

- Varying the ORTHONORMALITY and TOLERANCE values in an Excitation calculation

- Varying the linearscaling parameters

- Using diffuse functions

- Using the Dependency key

- Applying the ZORA relativistic corrections for molecules containing heavy nuclei

- Using an asymptotically correct XC potential such as LB94 or SAOP

Excitation Input

You can perform a calculation of singlet-singlet and singlet-triplet excitation energies of a closed-shell molecule by supplying in the input file the block key EXCITATION. See the next sections for the calculation of excitation energies for open shell molecules.

```
EXCITATIONS
 EXACT &
   IRREP1 N1
   IRREP2 N2
 SUBEND
```

```
 DAVIDSON &
   IRREP3 N3
   IRREP4 N4
 SUBEND
 ALLOWED
 ONLYSING
 ONLYTRIP
 CDSPECTRUM
 ANALYTIC
 VELOCITY
 LOWEST nlowest
 VECTORS nvect
 TOLERANCE tol
 ORTHONORMALITY orth
 ITERATIONS niter
 { Option }
 { Option }
  ...
End
```

Several options can be addressed with subkeys in the data block. This functionality is based on TDDFT and consequently has a different theoretical foundation than the SCF techniques described elsewhere in this User's Guide. Two possible ways are available to solve the eigenvalue equation from which the excitation energies and oscillator strengths are obtained, of which the iterative Davidson procedure is the default. In this case, the program needs to know how many excitation energies are needed per irrep, what accuracy is required, and what type of excitation energies are required (singlet-singlet or singlet-triplet). Suitable defaults have been defined for all of these. Each of these points is discussed below.

`Exact diagonalization vs. iterative Davidson procedure`

The most straightforward procedure is a direct diagonalization of the matrix from which the excitation energies and oscillator strengths are obtained. Since the matrix may become very large, this option is possible only for very small molecules. It can be activated by specifying the word EXACT as one of the subkeys in the Excitations data block. The default is the iterative Davidson method. A few of the lowest excitation energies and oscillator strengths are then found within an error tolerance. An advantage of the EXACT option is that additional information is produced, such as the Cauchy coefficients that determine the average dipole polarizability. The EXACT option not be used in unrestricted calculations.

`Singlet versus triplet`

By default, the singlet-singlet and singlet-triplet excitation energies are both calculated. The singlets are handled first, then the corresponding triplet excitation energies. One can skip one of these two parts of the calculation by specifying either ONLYSING or ONLYTRIP as a subkey in the data block.

In case of a calculation including spin-orbit coupling one can not separate the singlet-singlet and singlet-triplet excitations. The subkeys ONLYSING and ONLYTRIP are misused in this case to do a spin-restricted calculation, or a spin-polarized calculation, respectively. One should in fact only use the results of the spin-polarized calculation.

`Dipole-allowed versus general excitations.`

If you are interested in the optical absorption spectrum, you may not want to compute singlet-triplet excitation energies, nor singlet-singlet excitation energies which, by symmetry, have zero oscillator strengths. This subkey should not be used in case of spin-orbit coupling. The subkey ALLOWED tells ADF to treat only those irreducible representations for which the oscillator strengths will be nonzero. Of course, the oscillator strengths may still be negligibly small. The ALLOWED subkey automatically implies ONLYSING. The simplest, fastest, and recommended way to obtain information about the ten lowest dipole-allowed

excitation energies would be:

```
EXCITATIONS
 ALLOWED
 LOWEST 10
END
```

CDSPECTRUM

If the subkey *CDSPECTRUM* is included the rotatory strengths for the calculated excitations are calculated, in order to simulate Circular Dichroism (CD) spectra [80,81]. Interesting for chiral molecules. This subkey should not be used in case of spin-orbit coupling. For accuracy reasons you should also use the subkey *ANALYTIC* in the block key EXCITATIONS, otherwise the results may be nonsense.

ANALYTIC

If the subkey *ANALYTIC* is included the required integrals for the CD spectrum are calculated analytically, instead of numerically. Only used in case of CD spectrum.

Velocity

If the subkey *VELOCITY* is included ADF calculates the dipole-velocity representation of the oscillator strength. If applicable (use of subkey *CDSPECTRUM*) the dipole-velocity representation of the rotatory strength is calculated. Default the dipole-length representation of the oscillator strength and rotatory strength is calculated.

Which excitation energies and how many?

The user can specify how many excitation energies per irrep should be calculated. If no pertaining input is available the program determines these numbers from the smallest differences between occupied and virtual Kohn-Sham orbital energies. By default it looks at the 10 lowest orbital energy differences. This number can be modified, by specifying inside the Excitation block key, for example:

```
LOWEST 30
```

One should be aware that this procedure does not guarantee that the lowest 10 (or 30) excitation energies will actually be found, since the orbital energy difference approximation to the excitation energy is rather crude. However, if the program decides on the basis of this procedure to calculate 4 excitation energies in a certain irreducible representation, these 4 excitation energies are certainly the lowest in that particular irrep.

The user has more control when the number of excitations per irrep is explicitly specified within the EXCITATION block key by the Davidson subkey:

```
DAVIDSON &
 E'' 5
 T1.u 2
SUBEND
```

The DAVIDSON sub key is a general (simple or block type) subkey. For usage as block type it must, be followed by the continuation code (&). Its data block may contain any number of records and must end with a record SUBEND. In the subkey data block a list of irreps, followed by the number of requested excitation energies is specified. Note that the irrep name may not be identical to the usual ADF name. For example E'' is called EEE in ADF. The Excitation code will skip an irrep if the label is not recognized. For multidimensional irreps, only the first column is treated, because the other would produce identical output. This implies that the oscillator strengths for E-irreps have to be multiplied by 2 and the oscillator strengths for T-irreps by 3.

The EXACT subkey, mentioned already above, can also be used as a block type subkey to treat only a few

irreps instead of all. The number of excitation energies does not have to be specified then.

`Accuracy and other technical parameters`

A summary of technical parameters with their defaults is:

```
EXCITATIONS
 VECTORS 40
 TOLERANCE 1e-6
 ORTHONORMALITY 1e-8
 ITERATIONS 200
END
```

VECTORS: the maximum number of trial vectors in the Davidson algorithm for which space is allocated. If this number is small less memory will be needed, but the trial vector space is smaller and has to be collapsed more often, at the expense of CPU time. The default if usually adequate.

TOLERANCE: specifies the error tolerance in *the square* of the excitation energies in hartree units. The default is probably acceptable but we recommend that you verify the results against a stricter default (e.g. 1e-8) for at least a few cases.

ORTHONORMALITY: the Davidson algorithm orthonormalizes its trial vectors. Increasing the default orthonormality criterion increases the CPU time somewhat, but is another useful check on the reliability of the results.

ITERATIONS: the maximum number of attempts within which the Davidson algorithm has to converge. The default appears to be adequate in most cases.

Excitation energies for open-shell systems

Excitation energies can be obtained for open-shell systems in a spin-unrestricted TDDFT calculation [154]. This can not be used in case of spin-orbit coupling. To perform an open-shell TDDFT calculation one just needs to do an unrestricted SCF calculation and use the EXCITATION keyword. Presently the excitation energies can only be found with Davidson's procedure.

The printed symmetry in the output in TDDFT calculations is actually the symmetry of transition density. For closed-shell systems, the symmetry of the excited state is the same as the symmetry of the transition density, while for open-shell systems, the symmetry of the excited states is the direct product between the symmetry of the transition density and the ground state symmetry. Note that the ground state symmetry of an open shell molecule is not necessarily A1.

As for the spin-state, the general rule is that if the excited state mainly results from transitions from the singly occupied orbitals to virtual orbitals or from fully occupied orbitals to the singly occupied orbitals, the spin state of the excited state should roughly be the same as that of the ground state. However, if the excited state mainly comes from transitions from fully occupied orbitals to virtual orbitals, the spin state of the excited state are usually a mixture since TDDFT can only deal with single excitations within adiabatic approximation for the XC kernel [155]. Sometimes we just suppose the spin state of this kind of excited states to be the same as that of ground state [154]. In the MO $\rightarrow$ MO transitions part for the excitations of the output file, the spin of each molecular orbitals are also specified to help assign the spin state of the excited states. The transitions are always from α spin-orbital to α spin-orbital or from β spin-orbital to β spin-orbital.

Spin-flip excitation energies

Spin-flip excitation energies [156,157] can only be obtained in a spin-unrestricted TDDFT calculation. This can not be used in case of spin-orbit coupling. At present, the spin-flip excitation energies can only be calculated with Tamm-Dancoff approximation (TDA) [158] and Davidson's method.

To calculate spin-flip excitation energies, one must specify two keys:

```
SFTDDFT
```

and

```
TDA
```

anywhere in the input file in addition to the EXCITATION block keyword.

In spin-flip TDDFT, the XC kernel can be calculated directly from the XC potential. To use the LDA potential for the XC kernel, which roughly corresponds to the ALDA in ordinary TDDFT, one must specify the key

```
FORCEALDA
```

anywhere in the input file. Only calculations using the LDA potential in the SCF are fully tested. Using other GGA potentials in the SCF and using the FORCEALDA key at the same time may introduce unreasonable results, while using LB94 or SAOP potential in the SCF without the FORCEALDA key may give unstable results.

For open-shell molecules, spin-flip transition can result in transition to the ground state with a different S_z value, while the symmetry of the transition density is A1. The excitation energy of this transition should be zero and this can be used to test the reliability of spin-flip TDDFT.

The symmetry of the excited states can be determined in the same way as that in spin-unrestricted TDDFT calculations. As for the spin state, similar to that in the spin-unrestricted TDDFT calculations, some states may be more or less pure spin states, others may just be mixtures. The users can interpret the excited state through the transitions that contribute to this state. Note that the transitions are always from α spin-orbital to β spin-orbital in spin-flip calculations, or from β spin-orbital to α spin-orbital.

Core Excitation energies

The key MODIFYEXCITATION can be used to reduce the computational costs of core excitation energies, by allowing only selected occupied orbitals and or selected virtual orbitals in the TDDFT calculations. This can also be used in case of spin-orbit coupling. In this scheme the complete one-electron excited state configuration space is reduced to the subspace where only the core electrons are excited, see Stener et al. [169]. In the actual implementation this is done by artificially changing the orbital energies of the uninteresting occupied orbitals to a large negative value (default -1d6 hartree), and by by artificially changing the orbital energies of the uninteresting virtual orbitals to a large positive value (default 1d6).

```
MODIFYEXCITATION
   UseOccRange elowocc ehighocc
   UseVirtRange elowvirt ehighvirt
   UseOccupied
      irrep orbitalnumbers
      irrep orbitalnumbers
      ...
   SubEnd
   UseVirtuaL
      irrep orbitalnumbers
      irrep orbitalnumbers
      ...
```

```
    SubEnd
    SetOccEnergy esetocc
    SetLargeEnergy epsbig
    UseScaledZORA
end
```

`UseOccRange elowocc ehighocc`

Use only occupied orbitals which have orbital energies between `elowocc` and `ehighocc`.

`UseVirtRange elowvirt ehighvirt`

Use only virtual orbitals which have orbital energies between `elowvirt` and `ehighvirt`.

`UseOccupied`

Use only the occupied orbitals for which are specified.

`UseVirtual`

Use only the virtual orbitals for which are specified.

`irrep`

The name of one of the irreducible representations (not a subspecies) of the point group of the system. See the Appendix for the irrep names as they are used in ADF.

`orbitalnumbers`

A series of one or more numbers: include all numbers of the orbitals that are to be used. In an unrestricted calculation the same numbers are used for the spin-α orbitals and the spin-β orbitals.

`SetOccEnergy esetocc`

All occupied orbitals that have to be used will change their orbital energy to `esetocc`. In practice only useful if one has selected one occupied orbital energy, and one want to change this to another value. Default: the orbital energies of the occupied orbitals that are used are not changed.

`SetLargeEnergy epsbig`

The orbital energies of the uninteresting occupied orbitals are changed to -`epsbig` hartree, and the orbital energies of the uninteresting virtual orbitals are changed to `epsbig` hartree (Default: `epsbig` = -1d6 hartree).

`UseScaledZORA`

Use everywhere the scaled ZORA orbital energies instead of the ZORA orbital energies in the TDDFT equations. This can improve deep core excitation energies. Only valid if ZORA is used. Default: use the unscaled ZORA orbital energies.

Excitation energies and Spin-Orbit coupling

Starting from the ADF2006.01 version in ADF the relativistic TDDFT formalism, including spin-orbit coupling, is implemented for closed-shell molecules with full use of double-group symmetry [182]. This relativistic time-dependent density-functional theory (TDDFT) is based on the two-component zeroth-order regular approximation (ZORA) and a noncollinear exchange-correlation (XC) functional. This two-component TDDFT formalism has the correct nonrelativistic limit and affords the correct threefold degeneracy of triplet excitations.

In case of a calculation including spin-orbit coupling one can not separate the singlet-singlet and singlet-triplet excitations. The subkeys ONLYSING and ONLYTRIP of the key EXCITATIONS are misused in this case to do a spin-restricted calculation, or a spin-polarized calculation, respectively. One should in fact only use the results of the spin-polarized calculation, which is based on the noncollinear exchange-correlation (XC) functional.

To perform a spin-orbit coupled TDDFT calculation one just needs to do a spin-orbit coupled SCF calculation and use the EXCITATION keyword.

The contribution to the double group excited states in terms of singlet and triplet single group excited states can be estimated through the inner product of the transition density matrix obtained from two-component and scalar relativistic TDDFT calculations to better understand the double group excited states [183]. In order to get this analysis one needs to perform a scalar relativistic TDDFT calculation of excitation energies on the closed shell molecule first, and use the resulting TAPE21 as a fragment in the spin-orbit coupled TDDFT calculation of excitation energies, including the keyword STCONTRIB (Singlet and Triplet CONTRIButions):

```
STCONTRIB
```

Resonance Raman

According to a the time-dependent picture of resonance-Raman scattering the relative intensities of RR scattering cross sections are, under certain assumptions, proportional to the square of the excited-state energy gradients projected onto the ground-state normal modes of the molecule (see Ref. [202]).

Such excited-state gradients can be computed numerically by ADF's VIBRON module, which is invoked by selecting the VIBRON runtype in the GEOMETRY block key, the use of the VIBRON block key and the EXCITATION block key:

```
GEOMETRY
  VIBRON
END
```

```
VIBRON
 NMTAPE filename
 RESRAMAN
 {...
 ..}
END
```

```
EXCITATIONS
  LOWEST nlowest
END
```

The VIBRON module always requires an EXCITATIONS input block, in which the total number of excited states to be calculated must be specified. NMTAPE is the only obligatory keyword for the VIBRON module. It specifies the name of a TAPE21 file from a previous frequency calculation. This TAPE21 file is needed to read the normal modes w.r.t. which the derivatives are computed. I.e., a separate frequency calculation must be carried out first. The second subkeyword RESRAMAN invokes the resonance Raman calculation.

Resonance Raman for several excited states

The numerical evaluation of resonance Raman intensities has the advantages that

- Relative Intensities can be computed for several excited states at a time, since all excitation energies are determined simultaneously.
- Intensities can be computed for a selected no. of modes.

The intensities calculated for two different states cannot directly be compared, since the excited-state gradients only provide relative intensities for each excited state in resonance. For RR intensities from different excited states, also other quantities play a role. The most important one is the transition dipole moment to the excited state in resonance, which enters the intensity expression with to the fourth power.

Restrictions: (avoided) crossings between excited-states

The numerical calculation of excited-state gradients has a number of advantages, but also a possible problem: If the step size is chosen too large, or if there are close-lying excited-states, then the order of the excited states can change. For such cases, the excited-state gradient method to estimate relative RR intensities is not reliable: If states with different electronic character (but of the same symmetry) are close in energy, this will cause an avoided crossing. If the numerical derivatives are, in this case, computed w.r.t. the adiabatic states, they will probably not reflect the true situation. Especially if the coupling matrix elements between the two excited states is small, the spectroscopic properties often behave as if there is no avoided crossing, i.e., according to the diabatic states. Such cases should be handled with extreme care, since it is often not possible in advance to see whether the adiabatic or the diabatic picture should be invoked.

Because of the possible (avoided) crossings the user must make sure that always enough excited states are calculated to include the state(s) of interest. E.g., if the resonance Raman intensities are required for the first excited state, also some higher excited states have to be included in the excitation calculation, as the first excited state at the ground-state equilibrium might be higher in energy for displaced structures.

Restrictions: results not trustworthy for higher excited states

Users should be aware of another technical point: Excited states are usually calculated from a Davidson diagonalization procedure, i.e., only a small number of eigenvalues and eigenvectors describing the lowest excitations are obtained. During finite displacements, some of the higher calculated states might leave the calculated energy window, while others enter it. Hence, the character of some of the higher calculated states can change. In such a case, the numerical differentiation based on a (simple) diabatic pictures will fail for the higher states, since no mapping between the excited states for reference (equilibrium) and displaced structure can be carried out.

The solution is rather simple: Users should always ask for more excited states than they are actually interested in, and discard the data for higher states, in particular for those which could not successfully be mapped for displaced structures (look for messages in the output like 'State No. X cannot be expressed in terms of reference states').

For advanced users it should be mentioned that it is possible to set an energy window within the range of states calculated, and only the states within this energy window will be taken into account in the evaluation. See the subkey ELTHRESH and EUTHRESH of the block key VIBRON.

Furthermore, it is possible to pick out certain states from this energy window, and only perform the mapping (and diabatization, if requested) and differentiation for them. See the subkey SELSTATE of the block key VIBRON.

Advanced Restarts

In some cases, it only becomes obvious which states have to be included in a (simple) diabatization after excitation energies for all displaced structures are calculated. Therefore, the selected states and the energy window settings can also be adjusted in a restart (with the usual restart key) after all single-point calculations are done. However, this is only possible if all raw data are saved to TAPE21, which might be an enormous amount of data.

Therefore, the user has to specify the subkeyword SAVERAWDAT of the key VIBRON in the production run, and the subkeyword USERAWDAT of the key VIBRON in the (evaluation) restart. Such a restart does not invoke any new SCF and will, therefore, typically only take a couple of seconds or minutes. If SAVERAWDAT is not specified, restarts are still possible, but the energy window cannot be adjusted differently, and no new state selection can be performed.

Resonance Raman Input options

A number of options are available for the VIBRON module, most of which are for special applications. All the options mentioned below have to appear in the VIBRON block. The VIBRON module always requires an EXCITATION input block, in which the total number of excited states to be calculated must be specified.

```
VIBRON
 NMTAPE filename
 RESRAMAN
 {DISPTYPE disptype}
 {STPSIZE stpsize}
 {ONLYSYM}
 {NOTONLYSYM}
 {DOMODES list}
 {DONTMODES list}
 {DSCHEME dscheme}
 {EUTHRES euthres}
 {ELTHRES elthres}
 {SELSTATE list}
 {SAVERAWDAT}
 {USERAWDAT}
END
```

The most important ones in connection with RR calculations are:

`NMTAPE filename`

NMTAPE is the only obligatory keyword for the VIBRON module. It specifies the name of a TAPE21 file from a previous frequency calculation. This TAPE21 file is needed to read the normal modes w.r.t. which the derivatives are computed. I.e., a separate frequency calculation must be carried out first.

`RESRAMAN`

The second subkeyword RESRAMAN invokes the resonance Raman calculation.

`DISPTYPE disptype`

Select type of displacement steps; possible values are:
- `MASSWE`: steps in terms of mass-weighted normal mode vectors [default]

- CARTES: steps in terms of cartesian normal mode vectors
- REDUCE: steps in terms of reduced normal modes
- ENERGY: steps in terms of expected energy change (according to harmonic approximation)
- ZPELEV: like energy, but energy is expressed in ZPE units

STPSIZE stpsize

Sets the step size for the numerical differentiation in the default unit for the given DISPTYPE

ONLYSYM

Calculate derivatives only for totally symmetric modes (this is useful since this RR estimate only holds for Franck-Condon type Raman scattering, which is zero for non-symmetric modes). This option is ON per default in RR calculations, it can be switched off with the key NOTONLYSYM.

DOMODES list

Calculate derivatives only for the normal modes with numbers mentioned in list.

DONTMODES list

Calculate derivatives for all normal modes except the ones with numbers mentioned in list.

DSCHEME dscheme

The type of differentiation to be used can be set. Three different values for dscheme are available:

ELCHAR

A simple diabatic picture in which adiabatic states are mapped to the adiabatic states for the references structure based on a maximum transition density overlap criterion [default].

EIGVEC

A diabatic picture in which a diabatization is carried out as explained in Ref. [203]. I.e. the energies used here are the diagonal elements of the potential energy matrix for the nuclear Schrödinger Equation.

ADIABS

The adiabatic picture; can only be used if the symmetry of the excited states is supplied from a separate calculation, since the VIBRON module cannot check which states are allowed to cross as no symmetry is used in the excitation calculations. Consult the ADF-VIBRON manual [204] for information.

ELTHRESH elthresh
EUTHRESH euthresh

For advanced users it is possible to set an energy window within the range of states calculated, and only the states within this energy window will be taken into account in the evaluation.
- elthresh: lower bound in eV, default 0 eV.
- euthresh: upper bound in eV, default 1.0E10 eV.

SELSTATE list

For advanced users it is furthermore possible to pick out certain states from the energy window, and only perform the mapping (and diabatization, if requested) and differentiation for them. Here list includes the number (in ascending excitation energy) of the excited state at the reference (equilibrium) structure.

SAVERAWDAT

All raw data are saved to TAPE21, which might be an enormous amount of data. The selected states and the

energy window settings can then be adjusted in a restart (with the usual restart key) and the inclusion of the subkey USERAWDATA after all single-point calculations are done.

USERAWDAT

All raw data are read from a previous calculation. The selected states and the energy window settings can now be adjusted. You need to invoke the usual restart key. Such a restart does not invoke any new SCF and will, therefore, typically only take a couple of seconds or minutes. If SAVERAWDAT is not specified in the previuous calculation, restarts are still possible, but the energy window cannot be adjusted differently, and no new state selection can be performed.

Applications of the Excitation feature in ADF

It may be useful to consult the following (early) applications of the Excitation feature in ADF:

1. For excitation energies based on *exact* XC potentials: [82]
2. Calculations on Free Base Porphin: [83]; calculations on metal-porphyrins: a series of papers by Rosa, Ricciardi, Baerends, e.g. [84,85].
3. Calculations on MnO4-, Ni(CO)4 and Mn2(CO)10: [86]
4. Calculations on M(CO)5 (M=Cr, Mo, W), using the scalar ZORA relativistic approach: [87]
5. Excitation energies of open-shell molecules: [154,157]
6. Calculations on $[PtCl_4]^{2-}$, $[PtCl_4]^{2-}$, and $[PtCl_4]^{2-}$, using the ZORA relativistic approach including spin-orbit coupling: [183]
7. For details regarding the (near linear scaling and parallelized) implementation, please check Refs.[71,88]

Input description for the Response functionality

The calculation of frequency-dependent (hyper)polarizabilities and related properties is activated with the block key RESPONSE

```
RESPONSE
END
```

In this example only the *zz* component of the dipole polarizability tensor is calculated, at zero frequency. The orientation of the molecule is therefore crucial. Be aware that the program may modify the orientation of the molecule if the input coordinates do not agree with the symmetry conventions in ADF!
(This calculation could equivalently be done through a finite field method).

The impact of various approximations on the quality of computed polarizabilities has been studied in, for instance, Refs. [74,82,89]. If you are new to this application field, we strongly recommend that you study a few general references first, in particular when you consider hyperpolarizability calculations. These have many pitfalls, technically (which basis sets to use, application of the DEPENDENCY key) and theoretically (how do theoretical tensor components relate to experimental quantities, different conventions used). Please, take a good look both at ADF-specific references [75-77,90] and at general references related to this subject: Refs. [91-93], the entire issues of Chem.Rev.**94**, the ACS Symposium Series #628, and further references in the ADF-specific references.

Let's have a look at the available subkeys in the Response data block. (Not all of them should be used at the same time!)

```
RESPONSE
 ALLCOMPONENTS
 HYPERPOL LaserFreq
```

```
 DYNAHYP
 Nfreq Nfreq
 FrqBeg FirstFreq
 FrqEnd LastFreq
 [Optional Frequency/Energy Unit]
 ALLTENSOR
 DynaHyp
 Quadrupole
 Octupole
 VANDERWAALS NvanDerWaals
 RAMAN
 OPTICALROTATION
END
```

`Entire tensor or only one component`

You specify the AllComponents subkey to get the entire polarizability tensor, instead of just the *zz* component.

`Frequencies or wavelengths`

Instead of performing the calculation at zero frequency (which results in the *static* polarizability), one can specify an even-spaced sequence of frequencies, using the subkeys Nfreq, FrqBeg, and FrqEnd with obvious meaning. The (first and last) frequency values are by default in eV. This can be changed into Hartree units (a.u.) or in wavelengths (angstroms) by typing HARTREE or ANGSTROM on a separate line within the RESPONSE block, instead of [Optional Frequency/Energy Unit].

`Hyperpolarizabilities`

The first hyperpolarizability tensor b is calculated (in atomic units in the 'theoreticians convention', i.e. convention T=AB in Ref. [92]) if the subkey hyperpol is present with a specification of the laser frequency (in hartree units). If also the subkey allcomponents is specified, all components of the hyperpolarizability tensor will be obtained.

As mentioned before, by default only the static dipole hyperpolarizability tensor is computed. If one is interested in the frequency-dependent hyperpolarizability, the input could look like:

```
RESPONSE
 ALLCOMPONENTS
 HYPERPOL 0.01
 DYNAHYP
END
```

The subkey dynahyp has to be added and the main frequency ω has to be specified in Hartrees after the subkey hyperpol. In the output all nonzero components of the tensors governing the static first hyperpolarizability, second harmonic generation, electro-optic pockels effect, and optical rectification are printed.

Note: Second hyperpolarizabilities are currently not available analytically. Some can however be obtained by calculating the first hyperpolarizability in a finite field.
The effect of using different DFT functionals (LDA, LB94, BLYP) on hyperpolarizabilities in small molecules has been investigated in [77].

`Higher multipole polarizabilities`

Instead of just calculating the dipole-dipole polarizability, one may address the dipole-quadrupole, quadrupole-quadrupole, dipole-octupole, quadrupole-octupole, and octupole-octupole polarizability tensors. These can all be calculated in a single run, using the subkey alltensor. If only quadrupole-quadrupole or

octupole-octupole tensors are needed, the subkey quadrupole or octupole should be used.

`Accuracy and convergence`

```
RESPONSE
 erralf 1e-6
 erabsx 1e-6
 errtmx 1e-6
END
```

The subkeys erralf, erabsx, errtmx determine the convergence criteria for a polarizability calculation. The strict defaults are shown. It is rarely necessary to change the defaults, as these are rather strict but do not lead to many iterations.

`Dispersion coefficients`

Simple dispersion coefficients (the dipole-dipole interaction between two identical molecules, the C_6 coefficient) are calculated in a single ADF calculation. General dispersion coefficients are obtained with the auxiliary program DISPER, which uses two output files (file named TENSOR) of two separate ADF runs as input. See the Analysis and the Examples documents.
To get the dispersion coefficients one has to calculate polarizabilities at imaginary frequencies between 0 and infinity. The ADF program chooses the frequencies itself. The user has to specify the number of frequencies, which in a sense defines the level of accuracy, as an argument to the subkey VanDerWaals.

```
RESPONSE
 ALLCOMPONENTS
 VANDERWAALS 10
END
```

Ten frequencies is reasonable. In the example only dipole-dipole interactions are considered. If alltensor is specified, higher dispersion coefficients are also calculated. This ADF calculation generates a file with name TENSOR, which contains the results of multipole polarizabilities at imaginary frequencies. This TENSOR file has to be saved. Similarly, the TENSOR file for the second monomer has to be saved. The files have to be renamed to files 'tensorA' and 'tensorB' (case sensitive) respectively. Then the program DISPER has to be called in the same directory where the 'tensorA' and 'tensorB' files are located. DISPER needs no further input. See the Analysis document.

`Raman scattering`

Raman scattering intensities and depolarization ratios for all molecular vibrations at a certain laser frequency can be calculated in a single run. The run type must be Frequencies, which is arranged with the geometry key.

```
GEOMETRY
  FREQUENCIES
END
```

The RESPONSE key is used to specify that Raman intensities are computed.

```
RESPONSE
  RAMAN
END
```

In this example the static Raman scattering is calculated ($\omega = 0$). This type of calculation is very similar to an IR intensity calculation. In fact, all IR output is automatically generated as well. At all distorted geometries the dipole polarizability tensor is calculated. This is very time-consuming and is only feasible for small molecules.

There are a few caveats:
- Numerical integration accuracy must be high
- A calculation in which only a subset of the atoms is displaced is not possible for Raman calculations.
- For good results, a well converged (with the same basis and functional) equilibrium geometry must be used.

Because of this last point, it is wise to always start the RAMAN calculation with a TAPE13 restart file from a previous geometry optimization with the same basis, accuracy parameters, and density functional.

`OPTICALROTATION`

With the subkey OPTICALROTATION the (frequency dependent) optical rotation [80,94] will be calculated. For correct calculations one should calculate the entire tensor (see also the subkey ALLCOMPONENTS), which is done automatically.

Analysis options for TDDFT implementation (excitation energies and polarizabilities)

Several options are available to obtain more detailed results than a few bare numbers for excitation energies, oscillator strengths, transition dipole moments, and (hyper)polarizabilities. For a zero-order understanding of which occupied and virtual orbitals play an important in the polarizability or intensity of an absorption peak,

it may be useful to know the values of the dipole matrix elements between (ground-state) occupied and virtual Kohn-Sham orbitals. If these dipole matrix elements are large for a particular occupied-virtual orbital pair, then this pair is almost certainly of great importance for the whole spectrum or polarizability. This information can be obtained by specifying somewhere in the input file (but NOT inside the RESPONSE or EXCITATION block keys):

```
DIPOLEMAT
```

Time-dependent Current DFT

The time-dependent current-density-functional (TDCDFT) implementation is built entirely upon the normal TDDFT implementation. Therefore all general remarks that are made for the TDDFT part of the program are also valid for TDCDFT. Only the polarizability and excitation energies of closed shell molecules can be calculated with TDCDFT in the present implementation.

If TDCDFT is used together with the ALDA functional (NOVK option) it will give the same results for the polarizability and excitation energies as TDDFT in a complete basis set. TDCDFT in ADF by default uses the VK functional [160,161], since this is the only current dependent functional that is known presently. Many aspects of the functional are still unknown and the functional should therefore be used with caution. The user is referred to the references for more information on when the VK functional gives good results and when not.

For more information on the implementation and applications of the TDCDFT and the VK functional please read the references: [162-165]. For more details on the theory and implementation in ADF see: [166].

To activate TDCDFT and the VK functional one should add the following block key to the input file:

```
CURRENTRESPONSE
END
```

To calculate the polarizability the keyword Response can be used with the following options:

```
RESPONSE
 ALLCOMPONENTS
```

```
 Nfreq Nfreq
 FrqBeg FirstFreq
 FrqEnd LastFreq
 [Optional Frequency/Energy Unit]
END
```

The block key EXCITATION can be used with all of its options.

In default the VK functional will be applied where the NCT parameterization [167] is chosen for the transverse exchange-correlation kernel for the polarizability and singlet excitation energies (giving the best results for the systems studied so far). For triplet excitation energies the only available parameterization will be used [168]. This option is not tested much and the results are in general much worse than ALDA [166]. It is therefore suggested that VK is not used to calculate triplet excitation energies.

In the output the polarizability tensor (in case of an ALLCOMPONENTS calculation) has a different shape, the results are printed in the more intuitive order x, y, z, instead of y, z, x that the TDDFT implementation uses.

The following subkeys are available within the datablock of CURRENTRESPONSE

```
CURRENTRESPONSE
QIANVIGNALE
NOVK
END
```

QIANVIGNALE

The QV parameterization [168] will be used for the transverse exchange-correlation kernel instead of NCT.

NOVK

TDCDFT will be applied with the ALDA functional instead of the VK functional. In a complete basis this will give the same results as a TDDFT calculation.

ESR

Electron Spin Resonance properties are accessible with the keywords ESR and QTENS.

ESR is a block-type keyword that invokes calculation of the g-tensor [95] as well as the Nuclear Magnetic Dipole Hyperfine interaction (A-tensor) [96].

```
ESR
END
```

ESR is a block-type key, although it has not (yet) any data records to specify options or parameters. You can use the key in three situations:

- 1. In a Spin-Orbit spin restricted relativistic calculation the program will compute the G-tensor (Zeeman interaction). There must be exactly one unpaired electron (Kramer's Pair).
- 2. In a Spin-Orbit spin unrestricted relativistic calculation the program will compute the G-tensor and the Nuclear Magnetic Dipole Hyperfine interaction (A-tensor). The calculation must use the collinear approximation. There may be more than one unpaired electron.
- 3. If the Spin-Orbit feature is turned off, the calculation must be spin-*un*restricted and apply to an open-shell system. The program will compute the Nuclear Magnetic Dipole Hyperfine interaction (A-tensor).

In case (1), the program will also calculate and print the Nuclear Magnetic Dipole Hyperfine interaction, but

the terms due to the spin-polarization density at the nucleus are absent. Furthermore, if there is more than one unpaired electron, the computed results will simply be incorrect, without any warning from the program.

For the computation of the A-tensor, the Nuclear Magnetic Dipole Hyperfine interaction, an accurate evaluation of the spin-polarization density at the nucleus is important. This is best achieved in an all-electron calculation, avoiding any frozen core approximation.

Somewhat different ESR/EPR functionality is available from the CLGEPR (or briefly EPR) program which is described in the 'ADF Property Programs' documentation.

EFG

```
QTENS
```

This key activates the computation of the Nuclear Electric Quadrupole Hyperfine interaction. It can be applied to open-shell and to closed-shell systems. QTENS gives you the Nuclear Electric Quadrupole Hyperfine interaction (Q-tensor) [97]. The latter is directly related to the Electric Field Gradient (EFG) . The Q-tensor elements (in MHz) equal the the electric field gradient tensor elements (in a.u.) times 234.9647 times the nuclear quadrupole moment (NQM in barn units, 1 barn = 10-28m^2 = 10-24cm^2) and divided by 2I(2I-1), where I is the nuclear spin. The Nuclear Quadrupole Coupling Constant (NQCC) (in MHz) is the largest value of the principal values of the EFG (in a.u.) times 234.9647 times the nuclear quadrupole moment (in barn units). The electric field gradient tensor is printed next to the Q-tensor.

Electronic Configuration

The next few keys can be used to specify the electronic configuration. If you don't specify any such keys, certain defaults will apply. In principle, the program will (by default) attempt to find the lowest-energy spin-restricted (one-determinant) state. If SCF convergence is problematic the program may wind up at an excited state, by which (in this context) we mean a one-determinant state with a higher energy than some other one-determinant state with the same net spin polarization. In worse cases the program may fail to converge to any state at all. It is good practice to *always* verify which configuration you actually have computed.

When you specify a particular configuration and/or net charge and/or net spin-polarization of the system, the program will try to compute accordingly, even if the data have no physical or chemical meaning. The program has no knowledge about the existence of materials and will simply try to carry out what you tell it to do.

Spin: restricted vs. unrestricted

```
UNRESTRICTED
```

Specifies that spin-α and spin-β MOs may be spatially different and may have different occupation numbers. The default (absence of the key) is spin-restricted. The key has no argument. In the case of Spin-Orbit coupling it means that Kramer's symmetry does not have to be satisfied, in which case the key UNRESTRICTED should be used in combination with the key NONCOLLINEAR or COLLINEAR.

The unrestricted mode roughly doubles the computational effort. The actual numbers of spin-α and spin-β electrons respectively are controlled by the keys charge and occupations.Not e carefully, that using *only* the keyword unrestricted, without either Charge or Occupations (or both) would not result in any spin polarization. This implies that you would effectively perform a spin-restricted calculation, but with increased computational effort. Therefore, the program will check that in an unrestricted calculation at least one of the keys Charge and Occupations is applied.

The unrestricted feature is equivalent with, in *ab-initio* terminology, (Spin-)Unrestricted-Hartree-Fock (uhf); the N-particle wave function is a single determinant and not necessarily an eigenfunction of the spin operator S^2.

A *restricted* calculation implies that the (spatial) orbitals *and* the occupation numbers are identical for spin-α and spin-β.

The Fock operator, both in an unrestricted and in a restricted run, commutes with the spin operator Sz, but not (unless accidentally) with S^2. The obtained one-determinant wave function may for instance be a mixture of a singlet and a triplet state.

In an unrestricted calculation the expectation value of S^2 is now computed in ADF (note 29 in [98]). The implementation of an evaluation of S^2 is not quite trivial. DFT is essentially a one-particle formalism, so the S-operator for the n-particle system has to be written out in single-particle operators [99]. The equations used in ADF to calculate the expectation value of S^2 can be found in Szabo and Ostlund [100]. Note that the so called exact value $(S_{exact})^2$, which is printed in the ADF output, is defined as $(S_{exact})^2 = (|N_a - N_b|/2)(|N_a - N_b|/2+1)$, where N_a and N_b are the number of spin-α and spin-β electrons, respectively. The expectation value of S^2 is not calculated in a Spin-Orbit coupled calculation.

Molecules that have been calculated using the unrestricted formalism cannot be employed as fragments. ADF will abort when you attach the TAPE21 result file from an unrestricted calculation as a fragment file.

A fair approximation to a computation with unrestricted fragments can be achieved with the key FRAGOCCUPATIONS. See also the Examples.

Unrestricted and Spin-Orbit Coupling

In the case of Spin-Orbit coupling there are two ways to do spin-polarized calculations, either using the collinear approximation or the noncollinear approximation [101, 102]. Using the unrestricted feature in order to assign different numbers of electrons to a and b spin, respectively, cannot be applied as such, if one includes Spin-Orbit coupling, since the electrons are not directly associated with spin-α and spin-β. For the collinear and noncollinear approximation one should use symmetry NOSYM, and each level can allocate 1 electron. Note that with the key CHARGE one should only specify one value, namely the total charge. One should not specify the spin-polarization.

```
COLLINEAR
```

This key is only relevant in the case of Spin-Orbit coupling. The key has no argument. See also the key NONCOLLINEAR.

In the collinear approximation in each point in space the spin-polarization has the same direction (default is in the direction of the z-axis). Kramer's symmetry does not have to be satisfied. Symmetry used in the calculation should be NOSYM. The default direction of the spin-polarization can be overruled using the key SOUX (this key has no argument) for spin-polarization only in the direction of the x-axis, and the key SOUY (this key has no argument) for spin-polarization only in the direction of the y-axis. Both keys SOUX and SOUY are only relevant in the case of Spin-Orbit coupling in combination with the key COLLINEAR.

```
NONCOLLINEAR
```

This key is only relevant in the case of Spin-Orbit coupling. The key has no argument. See also the key COLLINEAR.

In the noncollinear approximation in each point in space the spin-polarization can have a different direction. Kramer's symmetry does not have to be satisfied. Symmetry used in the calculation should be NOSYM.

Net Charge and Spin polarization

The net charge of the molecule and the net spin polarization can be controlled with the key CHARGE.

```
CHARGE {NetQ {ab}}
```

`NetQ`

The net total charge of the molecule

`ab`

The net total spin polarization: the number of spin-α electrons in excess of spin-β electrons. Specification is only meaningful in a spin-unrestricted calculation. However, specification is not meaningful in an unrestricted Spin-Orbit coupled calculation using the (non-)collinear approximation.

If the key is used, the first value in the argument is assigned to netQ, the net total charge, and the second to ab. If the key is not used at all, default values apply. The default for the net total charge is the *sum of fragment charges: not necessarily neutral!!* The fragment charges are the net total charges that were used in the fragment runs; this information is stored in the fragment files.

The default spin polarization is zero.

An unrestricted calculation with ab=0 (for instance by not specifying charge at all) is in fact a restricted run: it should give exactly the same as the restricted calculation, but it will use more CPU time.

Orbital occupations: electronic configuration, excited states

With the key OCCUPATIONS you can specify in detail the assignment of electrons to MOs

```
OCCUPATIONS Options
  {irrep orbitalnumbers
  irrep orbitalnumbers
  ...
End }
```

`Occupations`

is a *general* key: it has an argument or a data block. If you want to use both, the continuation code (&) must be appended at the end of the argument.

`Options`

May contain Keeporbitals, Smearq, Freeze, or Steep:

`Keeporbitals=NKeep`

Until SCF cycle Nkeep electrons are assigned to MOs according to the Aufbau principle, using at each cycle the then current orbital energies of the MOs. Thereafter the KeepOrbitals feature is applied. As soon as this is activated the program will on successive SCF cycles assign electrons to the MOs that maximally resemble - in spatial form - those that were occupied in a 'reference cycle number'. The default for Nkeep is 20, except:
a) When orbital occupations for MOs are specified explicitly in the data block of the occupations key, these apply throughout.
b) In a Create run fixed occupations are derived from a database in the program.
c) When electron smearing is explicitly turned on by the user (see the Smearq option below) Nkeep is by

default 1,000,000 so the program will 'never' compare the spatial forms of MOs to determine the occupation numbers.
The 'reference cycle number' is by default the previous cycle, which will suppress jumps in the spatial occupations during the SCF development while at the other hand allowing the system to let the more-or-less-frozen configuration relax to self-consistency.

`Freeze`

Occurrence of this word in the option list specifies that the 'reference cycle number' will be the cycle number on which the KeepOrbitals feature is activated: during all subsequent SCF cycles the program will assign electrons to MOs that resemble the MOs of that specific SCF cycle. This may be used when the MOs of that cycle are already reasonably close to the final ones, and it will suppress unwanted step-by-step charge-transfers from occupied to empty orbitals that are very close in energy. By default this option is not active.

`Smearq=Smear1[,Smear2,Smear3,...,Smear10]`

Smear*N* is half the energy width (in hartrees) over which electrons are smeared out over orbitals that lie around the fermi level and that are close in energy. Smearing is a trick that may help when the SCF has problems converging. One should be well aware that the physical meaning of a result obtained with smeared occupations is unclear (to express it mildly). It may be useful to get over a hurdle in a geometry optimization.
By default the *initial* smear parameter is zero (i.e.: smearing is not applied). It is turned on automatically by the program when SCF convergence is found to be problematic, but only in an optimization-type application (simple optimization, linear transit, transition state) when the geometry is not yet converged.
You can rigorously prohibit any smearing by specifying it explicitly with value zero. More generally: specifying the smear parameter makes the program to apply it always, but always with the input-specified value.
When a comma-delimited list of values is specified, after SCF has converged, the next value from the list is picked and the SCF is continued. This way one can specify a list of gradually decreasing values to get sort of annealing effect. NOTE: No spaces are allowed when specifying a list of values for Smearq.

`Steep=Lambda[,Nmax]`

The occupation number for each orbitals are updated according to steepest-descent method (Ref: F. W. Averill and G. S. Painter, Phys. Rev. B **46**, 2498 (1992)). During an SCF cycle, the occupation number for each new orbital is initially determined by decomposing the old charge density with new orbitals. Then, the occupation numbers are modified so that the total energy of the system will decrease.
The Lambda parameter gives the coefficient for the charge transfer in 1/au unit. The second parameter, Nmax, is an additional limit for the amount of the charge transfer. Nmax would be useful for early steps of cycle when the Lambda parameter gives too large charge transfer. Too small Nmax results in irregular behavior in SCF convergence. In the case of difficult SCF convergence, you should make mixing and Lambda smaller. From our experience, Nmax=0.1 or 0.2 is usually OK.
This method should be used with turning off DIIS method (DIIS N=0), and the choice of the mixing parameter in SCF cycle is also important. This option is especially useful for systems with many quasi-degenerate orbitals around Fermi level. For instance, cluster models of surface systems usually suffer from dangling bonds and should be converged with this method. Note though that slow convergence is an intrinsic feature of this method so one should specify a large limit for the number of SCF cycles, say 500 or even 1000, depending on the cluster size.

`irrep`

The name of one of the irreducible representations (not a subspecies) of the point group of the system. See the Appendix for the irrep names as they are used in ADF.

`orbitalnumbers`

A series of one or more numbers: the occupation numbers for one-electron *valence* orbitals in that irrep. The orbitals are ordered according to their energy eigenvalue; higher states than those listed get an occupation number zero.

For degenerate representations such as the 2-dimensional E-representations or the 3-dimensional T-representations, you must give the *total* occupation, i.e. the sum over the partner representations; adf assigns each partner an occupation equal to the appropriate fraction of what appears here.

In an unrestricted calculation, two sequences of numbers must be specified for each irrep; the sequences are separated by a double slash (//). The first set of numbers is assigned to the spin-α orbitals, the second set to the spin-β orbitals. Note that this is not meaningful in an unrestricted Spin-Orbit coupled calculation using the (non-)collinear approximation, where one should use one sequence of occupation numbers for each irrep.

Notes about the occupations data block:

- When specifying electron configurations, all valence electrons in the calculation must be explicitly assigned to MOs
  and the block form of the OCCUPATIONS keyword must be used.
  In this context the concept *valence electrons* and hence *valence orbitals* is not necessarily identical to what you may normally assume to be the valence space of an atom or molecule.
  The meaning of *valence* is here strictly defined as whatever electrons are outside the frozen core.
  It depends therefore on the level of frozen core approximation applied in the calculation.
  This traces back to the Create runs in which the basic atoms were generated that are now used to build the molecule.
- When for some irrep there is a rather long list of occupation numbers, corresponding to *consecutive fully occupied* states, you can combine these numbers and enter their sum instead:
  adf knows the maximum occupation for an irrep, and when you put a larger number the program will split it up.
  For instance, if you give for the *p*-representation (in a single atom calculation):
  P 17 3
  adf will interpret this as
  P 6 6 5 3
  i.e. the occupation number 17 is interpreted as denoting two fully occupied p-shells and the remaining five
  electrons in the next higher shell.
  This example also illustrates how to specify an excited state: here we have defined a hole in the third p-shell.
- Fractional occupation numbers in input are allowed. For a discussion of the interpretation of fractional occupation numbers see [103]. The program even allows you (technically) to use a non-integer
  total number of electrons, whatever the physical meaning of such a calculation is.
- The data block of occupations is not parsed (see the section Interpretation of Input below).
  The program does not replace expressions by their value and it does not recognize constants or functions
  defined with the define key.
- In a Frequencies run the symmetry used internally in the program is nosym, irrespective of any Schönfliess
  symbol in the input file. As a consequence the program will recognize only the A representation
  (the only irrep in nosym), but not the representations belonging to the input point group symmetry.
  (The symmetry in the equilibrium geometry, defined by the input Schönfliess symbol, is used to enhance
  efficiency and stability in the construction of the matrix of Force constants).

Notes about the occupations options:

- When occupation numbers are explicitly defined via the block form of the OCCUPATIONS keyword, the Smearq option cannot be used.
- The aufbau principle does not determine or adjust the distribution of electrons over spin-α versus spin-β
  in an unrestricted calculation. This aspect is controlled by the key CHARGE and by any explicit occupations in
  the data block of occupations.
- When occupation numbers are not specified and no Smearing is specified either,
  the program will turn on smearing automatically when the SCF has serious convergence problems, in an attempt
  to overcome those problems, but only in a geometry optimization (including transition state, linear transit, etc.).
  If such happens the program restores the original situation (no smearing) at the start of each new SCF.
  In automatic smearing the smear parameter is initiated at 0.01 hartree and may be varied (by the program)
  between 0.001 and 0.1 hartree. The automatic use of smearing by the program can be
  prohibited by explicitly setting the smear option with value zero (Smearq=0).
- Smearing cannot be used in combination with the keeporbitals option.
  This option therefore also turns of *automatic* smearing in troublesome SCF 's during an optimization.

CHARGE vs. OCCUPATIONS

The contents of the data block of occupations, if used, defines the total number of valence electrons and hence the net total charge. In an unrestricted run it also defines the net spin polarization. If the key CHARGE is also used, the program will check that both specifications are consistent.

We strongly recommend to employ this and always specify the net total charge and spin polarization with charge whenever explicit occupation numbers are supplied with occupations, to that the program will check that your occupation numbers result in the total charge and spin polarization that you have in mind.

Create mode

In Create mode occupation numbers are predefined (see Appendix 2 Elements of the Periodic Table), and these are applied unless you specify occupations in input yourself. Conceivably this may result in a non-aufbau configuration. In Create mode the program always operates as if occupations were set in input.

Frozen core vs. pseudopotentials

Pseudopotentials are not supported. The frozen core approximation is automatic in a normal (Fragment mode) calculation and is defined by the basic atomic fragments. The data file used in the Create run specifies the frozen core for the atom, which is then used in all molecules that incorporate that atomic fragment.

Multiplet States

Calculations with adf yield results for one-determinant electronic states, which are not always the 'true' states of the molecule. The evaluation of the correct multiplet energies is not trivial in this approach; see the Theory document. The point is to evaluate a specific multiplet state as a linear combination of selected one-determinant functions, each computed in the field of the so-called Average-of-Configuration (AOC). Typically, in an open shell system, the AOC is the spin-restricted system in which all orbitals in the open

shell are degenerate and equally occupied. The AOC serves then as a fragment for the subsequent calculations, in which the different open shell orbitals are occupied differently by specifying the appropriate occupation numbers as explained below.

Important: in these follow-up calculations it is imperative that the results are obtained in the AOC field: no SCF convergence must be carried out, because we only want to assign the electrons differently, while keeping exactly the AOC orbitals. To achieve this, the follow-up calculations must use the keyword SCF, and the subkey iterations must be set to 0.

Since adf requires that the point-group symmetry matches not only to the nuclear frame but also to the electronic charge density and MO occupations, these calculations must run in a lower pointgroup symmetry. Often you will also want to run the modified calculations spin-*un*restricted. For an example, see the set of sample runs that come with the package and the discussion in the Examples document.

The calculation of the one-determinant states based on the AOC reference state is controlled with the key SLATERDETERMINANTS: . It is a *general* key; it can be used as a simple key and requires an argument then. It can also be used as a block key. For this particular key it is not correct to specify an argument *and* a data block.

```
SLATERDETERMINANTS file
```

When used as a simple key, the argument must be a file (including the path). The file must be an ASCII file containing data in the same format as you would supply in the data block when using the key as block type key, see below. All information on the file until the *eof* must be suitable for the data block, but no record 'end' on the file must be specified: only the *contents* of the data block.

The block format:

```
SLATERDETERMINANTS file
 title1
 irrep occups
 irrep occups
 ....
 subend
 title2
 irrep occups
 .....
 subend
 title3
 ....
 subend
 ....
end
```

Each 'title' functions as a subkey, but is otherwise an arbitrary string to label the resulting one-determinant calculation. Each such subkey block contains the occupation numbers for a single one-determinant calculation. It is necessary that the calculation uses the reference AOC run as its only fragment file. The occupations in the subkey blocks must be re-arrangements of the AOC open-shell electrons. In the Slaterdeterminants calculation you must explicitly specify the pointgroup symmetry in which you want to run; this must be a lower symmetry than the AOC one, otherwise you couldn't rearrange the open shell electrons. See the Theory document. An sample run is included in Examples document.

Each 'irrep occups' record specifies the occupations for the indicated irrep in the usual way (see for instance the occupations key). The irrep labels must correspond to the (lower) point group symmetry used in the slaterdeterminants calculation. Note that in an unrestricted calculations, occupations numbers must be given for both spins, using the double slash (//) to separate the occupations for spin-α and spin-β.

In this setup, the program will for each of the subkey blocks under the slaterdeterminants key execute an SCF calculation with only one cycle, i.e. no convergence, where the start-up field is the fragment field, i.e. the AOC field. So all one-determinant states in this calculation are evaluated in the AOC field. The resulting energies for the distinctly computed one-determinant states can then be combined to the desired multiplet values, corresponding to how the multiplet states are combinations of the one-determinant states.

Precision and Self-Consistency

The precision of a calculation is determined by

- The function sets (basis sets, levels of frozen core approximation, and fit sets for the computation of the Coulomb potential)
- Numerical integration settings in real space and in k-space
- Convergence criteria (for the SCF procedure and the geometry optimization)
- A few more items that are rather technical and usually irrelevant (these are not discussed here).

The fragments you attach determine, through the fragment files, the function sets. Since each fragment traces back to one or more Create runs, the employed data base files in the Create runs determine the finally employed function sets.

For convergence of the geometry optimization see the key GEOMETRY.

In this part we examine numerical integration and the SCF procedure.

Numerical Integration

Many integrals in adf are evaluated by numerical integration: Fock matrix elements, several terms in the (bonding) energy, gradients in geometry optimization, and so on. A sophisticated numerical integration procedure is used [104, 105]. It requires only one input parameter which determines the precision of numerical integrals and derives from that the number of integration points.

```
INTEGRATION accint
```

`accint`

A positive real number: the numerical integration scheme generates points and weights such that a large number of representative test integrals are evaluated with an accuracy of accint significant digits. The default for accint depends on the runtype: 4.0 for Single Point runs and simple Geometry Optimizations, including Linear Transits; 5.0 for Transition State searches; 6.0 for the computation of Frequencies; 10.0 in Create runs.

The *number* of integration points varies strongly with accint, and this determines to a large extent the computational effort. Decreasing accint from 4.0 to 3.0 for instance roughly halves the number of points (this depends somewhat on the molecule).

The defaults should yield good precision for the very large majority of applications. Lower values (3.0 or even 2.0) can be used if precision is not crucial and the purpose is to get an impression. We recommend that you experiment for yourself to get a feel for how results may vary in quality and computing time.

The default in Create mode is very large: 10.0. This is computationally no problem thanks to the simplicity of the single atom case, in particular due to the high symmetry. There is no reason to override the default integration settings when creating basic atoms.

Frequencies

The computation of *frequencies* should always be carried out with accint at least 4.0 to get results that make sense; many molecules, in particular those than contain metals, require higher precision, which is why the default is 6.0. The reason is that frequencies are computed by numerical differentiation of gradients computed in slightly displaced geometries. Obviously the *noise* in the gradients due to numerical integration errors should be small compared to the difference in gradients across the different geometries and since the latter are very close, a high numerical accuracy is required [106-108].

Self-adapting precision during optimizations

In Geometry Optimizations and Transition State searches, the gradients at the initial geometry may be quite large and in such case there is no need to apply the same high integration precision as may be required close to convergence when the gradients become very small. Therefore, adf tries to reduce the numerical integration accuracy during optimization to save time, and use the user-specified precision only in the final stage of the optimization.

The program starts with an initial value accfirst, to get an assessment of the (local) gradients. Subsequently it adjusts it according to the progress towards convergence. All values are kept between a lower and an upper bound: accmin and accmax respectively.

All three parameters, accfirst, accmin, accmax, are controlled by the key INTEGRATION.

```
INTEGRATION acc1 {acc2 {acc3}}
```

The simplest application, discussed above, specifies *one* value (acc1). This defines then both the upper bound (accmax) and the first value (accfirst). The lower bound accmin is by default 3.0 (adjusted internally to acc1, if that is lower); 3.5 in Transition State searches.

If *two* values are supplied, the smallest is taken for the lower bound, the larger for the upper bound. The value for the first cycle equals the upper bound.

If *three* values are supplied, the smallest is the lower bound, the largest the upper bound, the remaining value is used to start with.

More integration options

We've now only explained the normal, simple application of the Integration key, which we hope and expect is adequate for all your computations. In chapter 2.3 'More Options' additional details will be discussed. The distribution of points over space is internally regulated by quite a few parameters. Each of these parameters can be controlled in input. By default they depend on one another, and all of them depend on the main parameter accint. Advanced users may wish to experiment and override the default relations between the parameters.

You may also have rather non-standard applications where the default relations are less adequate. A thorough understanding of the underlying method is required to make a sensible choice for all parameters [105, 109].

SCF

The SCF procedure is regulated with keys that set the maximum number of iterations, the convergence criterion, and various items that control the iterative update method. Molecules may display wildly different SCF-iteration behavior, ranging from easy and rapid convergence to troublesome oscillations. We expect that the default settings take care of most cases, but one should realize that this is a difficult and tricky subject.

The user has a few (main) options to adapt the procedure to the situation at hand: simple damping or the DIIS procedure (Direct Inversion in the Iterative Subspace). Either of them can be combined with Level-Shifting.

At each cycle the density is computed as a summation of occupied orbitals (squared); the new density defines the new potential from which the orbitals are re-computed, et cetera, until convergence is reached. To speed-up convergence and to avoid non-convergent oscillatory behavior the values at the next iteration are constructed as a mixture of the computed new data and those used at the cycles before. This may involve only the previous cycle and is then called *damping*. Alternatively the DIIS procedure can be invoked, which is a generalization of damping to include more previous iterations.

Subkeys in the data block of the master key SCF control the aspects mentioned above. Each subkey is optional. Omission means the application of default values. Omission of the SCF key altogether implies defaults for all subkeys.

```
SCF
 Iterations Niter
 Converge SCFcnv { sconv2 }
 mixing mix
 diis {N=n} {OK=ok} {CX=cx} {CXX=cxx} {BFAC=bfac} {cyc=cyc}
 lshift vshift {Err=shift_err} {Cyc=shift_cyc}
End
```

`Niter`

The maximum number of SCF cycles allowed. In case of Geometry Optimizations it applies separately to each of the SCF procedures that are executed. Default is 50 (in Create mode: 100). The program executes at least one cycle, even if you specify a non-positive number.

`SCFcnv`

The criterion to stop the SCF updates. The tested error is the commutator of the Fock matrix and the P-matrix (=density matrix in the representation of the basis functions) from which the F-matrix was obtained. This commutator is zero when absolute self-consistency is reached. Convergence is considered reached when the maximum element falls below SCFcnv and the norm of the matrix below 10*SCFcnv. The default is 1e-6 (in Create mode: 1e-8).

`sconv2`

A second criterion which plays a role *a*) during geometry optimizations and *b*) when the SCF procedure has difficulty converging.
a) During an optimization the SCF convergence criterion is relaxed as long as the geometry has not yet converged. At the start-up geometry, and at the final geometry, the normal criterion (SCFcnv) is applied, at intermediate cycles the criterion is adjusted depending on how far the geometry has converged. sconv2 defines a minimum criterion: the actual criterion in effect will not be less than sconv2.
b) When in any SCF procedure the currently applicable criterion does not seem to be achievable, the program stops the SCF. When the secondary criterion (sconv2) has been met, only a warning is issued and the program continues normally. If the secondary criterion was not met either, the program terminates any further geometry optimizations, frequency steps, etc. You can prevent the program from terminating in such a case with the key ALLOW.
The default for sconv2 is 1e-3.

`Mix`

The relative weight of the new potential, as computed from the occupied orbitals, to be mixed with the potential that was used in the previous cycle, to define the potential for the next. Mixing is used only if, and as long as, the DIIS procedure (see below) is not operational. Default: 0.2.

For problematic systems that require strong damping, one should *decrease* the mix-parameter.

`DIIS`

The DIIS subkey specification(s) can be given to control the DIIS procedure. Each of these specifications is optional. Simple damping will be used during the first few cycles, until the DIIS procedure becomes operational. Two conditions must be satisfied for this: 1) at least two iterations must have been done anyway (to build up sufficient information for the DIIS to work at all) and 2) the error must be small enough; see however the cyc option below.
There have been claims in the literature that the DIIS should not be used until fair convergence has been reached. Our experience thus far does not indicate that this should be taken too seriously, except in special situations. To allow the user complete control, the start-up criteria can be set in input.

`N`

The number of expansion vectors used in the DIIS. The number of previous cycles taken into the linear combination is then n-1 (the new computed potential is also involved in the linear combination.
Default n=10. An input value smaller than 2 disables the DIIS.

`OK`

The DIIS starting criterion. The DIIS procedure is not invoked until a) the maximum commutator element is smaller than OK (default: 0.5) or *b*) a certain number of SCF cycles has been executed.

`Cyc`

The SCF cycle no. at which the DIIS will start irrespective of the OK value above. Default: 5.

`Cx`

An upper bound on linear combination coefficients as applied in the DIIS. As soon as any coefficient exceeds cx, all information about older cycles but the last two is discarded and the DIIS starts again to accumulate info from the current cycle on. The computed linear combination, with the large coefficient(s), is used for the next iteration, however. Default=5.0

`Cxx`

A second upper bound on the coefficients (should in principle be bigger than cx). When a coefficient exceeds cxx, the computed linear combination is not used for the next cycle, but simple damping is applied.

`Bfac`

A factor to bias the DIIS combination vector in favor of the new computed potential. Default=0 (no bias). A sensible alternative value, advocated in [110] is 0.2

`VShift`

The level shifting parameter. The diagonal elements of the Fock matrix, in the representation of the orbitals of the previous iteration, are raised by vshift hartree energy units for the virtual orbitals. This may help to solve convergence problems when during the SCF iterations charge is sloshing back and forth between different orbitals that are close in energy and all located around the Fermi level. Level shifting is not supported in the case of Spin-Orbit coupling. At the moment properties that use virtuals, like excitation energies, response properties, NMR calculations, will give incorrect results if level shifting is applied.

`Shift_err`

Specifies that level shifting will be turned off by the program as soon as the SCF error drops below a threshold; default value: 0. (Note that the default value has changed in ADF2004.01, previous value: 1e-2).

`Shift_cyc`

Specifies that level shifting is not turned on before the given SCF cycle number (for the start-up geometry); default value: 1.

Note1: very strong damping, i.e. a very small value of mix such as 1e-3, may not combine very well with the DIIS procedure. The reason is that with strong damping successive SCF cycles tend to be very similar and the vectors building up the DIIS space become linearly dependent. We recommend in difficult cases either to use a not too strong damping (mix=0.03) *or* to use strong damping while the DIIS is disabled (by setting n=0 for the DIIS subkey) during a limited number of SCF iterations, and then *restart* with DIIS activated and less stringent damping.

Note2: Another feature, *electron smearing*, may be used to overcome convergence difficulties. The idea is to distribute electron occupations fractionally over a few states around the Fermi level, by a pseudo-thermal distribution function. This aspect is controlled with the Smear option to the Occupations key. One should be aware that the applied distribution of occupations is not really an approximation to the finite-temperature case. In fact, the results are unphysical and one should not use the results as a meaningful outcome. The smearing trick is *only* to be used to overcome convergence difficulties. Having reached convergence with it, one should typically do a follow-up restart calculation without smearing, using the converged outcomes to hopefully get the thing to converge properly. A typical 'allowed' application is the usage of smearing during geometry optimizations, because the intermediate geometries are not relevant anyway and only a step towards the final results. By default, the program does *not* apply any smearing unless during a geometry optimization. See the Occupations key for more details.

Interpretation of Input

ADF has two special keys that regulate the specification and interpretation of numerical data in input. These keys, and related aspects, are convenient for the formatting of input.

The position of the interpretation keys in the input file is significant! Therefore, to avoid problems and misunderstandings: before supplying any numerical data, specify first (if at all) the keys units and define (see below).

Units of length and angle

Geometric lengths and angles are in units defined by:

```
UNITS
 length Angstrom / Bohr
 angle Degree / Radian
end
```

Angstrom and Bohr, respectively Degree and Radian, are recognized strings. Each of the subkeys is optional, as is the key UNITS itself. Defaults: Angstrom for lengths, and Degree for angles.

The position of the key UNITS in input is significant as regards the evaluation of expressions (see the paragraph on constants and functions below). In other respects its position plays no role. To avoid mistakes one should place units as early as possible in input (if at all).

Expressions

ADF supports the use of arithmetic *expressions*, *functions*, and *constants* to represent numerical data. This can be convenient for the input of, for instance, atomic positions when these would most easily be

represented in terms of 1/3, sin(360/5), et cetera. Using expressions and functions is easier, avoids the tedious typing of long decimal expansions and solves the question of precision (how many digits should be supplied?).

The standard arithmetic operands in Fortran (+ - * / **) can be applied in expressions, together with parentheses where suitable.

Blanks are allowed and ignored, but they are interpreted as separators, i.e. as denoting the end of an expression, whenever the part until the blank can be evaluated as a correct expression. For instance 3* 4 will be interpreted as 12, but 3 *4 will be interpreted as 3, followed by a character *, followed in turn by the number 4.

All numbers and results are interpreted and handled as being of type real, but whenever the result is a whole number (allowing for very small round-off) it will be recognized and accepted as an integer when such data is required.

Constants and functions

The user may define *constants* and *functions* in the input file, and apply them subsequently in expressions. The input file is read sequentially and *constants and functions must be defined before they can be used*.

The argument list of a function must be enclosed in parentheses and the arguments, if more than one, separated by commas.

The following functions are predefined in adf and can be used directly in input:

sin, cos, tan, asin, acos, atan, exp, log, sqrt, nint. Each of them has one argument. log is the natural logarithm (base *e*).

No constants are predefined.

The angular argument to the trigonometric functions cos, sin, tan is in the unit for angles as defined by units, *provided the unit has been set before it is applied*. For the result of the inverse trigonometric functions the same holds.

Constants and functions can be defined with the block key DEFINE:

```
DEFINE
 angle=54
 ab = sin(angle/3)
 s13 = 14*sqrt(2)
 func(x,y,z) = x*ab+y**2-y*z
end
```

The constants angle, ab, and s13 are defined together with a function func, using the predefined functions sin and sqrt. These can then be applied to assign values elsewhere in input.

In the example above, the constant angle is used in the definition of ab, and ab is used in turn to define func; these constructions are allowed because angle is defined before ab, and ab is defined before func.

The replacement of constants, functions, and other expressions by their numerical values may considerably increase the *length* of the input record, in particular when real values are being generated (by the parser) in the standard format E22.14. Take care that the resulting record does not exceed 80 characters. The program will abort or may run into an error if this is violated.

The input-reading routine applies the constants and functions wherever it is allowed to do so. To prevent any unwanted replacements in the input file you should avoid very short identifiers for constants and functions.

Warning example:

```
DEFINE
 A=3.18
 C=4.12
end
...
atoms
 C 0.00 1.05 -3.22
...
```

The program will apply the definition of the variable C and read:

```
DEFINE
 A=3.18
 C=4.12
end
...
atoms
 4.12 0.00 1.05 -3.22
...
```

Avoid single-character identifiers!

Strings

Quotes can be used to designate strings, i.e. (parts of) records which are not to be parsed for expressions, but which should be taken as they are. The quotes themselves are ignored, i.e. removed by the parser. Two consecutive quotes inside a string are interpreted to denote the (single) quote character as a part of the string.

Where does parsing apply?

Replacing pre-defined variables and expressions by their value is applied only to keys that carry numerical data. For example: atoms, define, units. However, it is *not* applied to keys that carry electronic occupation numbers.

Note that when parsing applies to a given key the whole record of the key (key + argument) and its data block are parsed. The parsing then applies to all items, even those that in themselves have no numerical meaning (for instance, the atom type names in the atoms data block are scanned and must of course then not be 'defined' as identifiers with a numerical value.

Constants vs. geometric parameters

Note carefully the difference between constants defined with define and identifiers that are used for atomic coordinates in the data blocks of atoms and geovar. Constants defined under define are merely symbols for, and exactly equivalent to, certain numerical values, whereas the coordinate identifiers carry implications such as the distinction between frozen and optimization coordinates. *Constants* affect only the input *after* their definition and the location of their definition in the input file is significant. Geometric *identifiers* only relate to the data blocks of atoms and geovar respectively and the relative order in which the keys atoms and geovar occur is irrelevant.

Restarts

Check-point file

When an adf calculation terminates abnormally - not controlled by the program itself, for instance after a core dump due to some bug - there will usually be a file TAPE13, which serves as a checkpoint file. tape13 can be used to restart the calculation at a point not too far before the fatal condition occurred. It contains only data for the restart, but none of the special analysis data on TAPE21 that would be useful for analysis, to serve as fragment file, etc.

TAPE13 is upgraded during the calculation but discarded upon normal termination, namely when all relevant information has been saved on TAPE21. At that point all info that would have been on TAPE13 is present on TAPE21. If you wish to keep tape13 anyway - for instance because you plan a restart after normal termination and don't intend to keep the substantially bigger TAPE21 - you must use the save key.

Upon normal (i.e. program-controlled) termination of a calculation, the TAPE21 result file can be used for restart purposes. When a crash occurs, however, chances are that TAPE21 has not correctly been closed and that its data structure is inconsistent: during the calculation large portions of TAPE21 are kept in memory rather than on file, and only at the point of final termination, all data is flushed to file.

General remarks

In all restart calculations a normal input file must be supplied (you can, for instance, simply take the original one), with a specification of the restart file added: the restart file does *not replace* the input file. From the program's point of view, it first reads the 'normal' input file and then inspects whether a restart file is present to replace some of the information read from input.

The concept of restarts in adf is rather simple and primarily directed at increasing computational efficiency by providing cost-expensive data. The continuation run is to a large extent independent from the one that generated the restart file. The runtype, the choice of density-functional and other features in the Hamiltonian, precision of numerical integration, thresholds on convergence, et cetera are all determined solely from the input file for the new run: no such data is read from the restart file. Most input items should, therefore, be supplied in the restart run again, even if it is a direct continuation of a previous calculation: omission implies using the standard defaults, which are not necessarily the settings of the calculation that generated the restart file.

Even the key ATOMS with the list of atomic coordinates must be supplied again: the program needs the information herein to deduce what fragments are used, which coordinates are free or frozen respectively in an optimization, etc. The coordinate *values* may be supplied with the restart file and these will then overwrite those specified in the input file.

Obviously, the two runs cannot be completely unrelated. To let the restart data make sense the runs should correspond to the same molecule (i.e. its general definition in terms of fragment building blocks). The program does not check all aspects related to this and certain abuses will therefore survive the internal tests, but will surely lead to some error later on: it is the user's responsibility to ensure that the restart data match the calculation one has in mind.

Interdependencies between data read from the restart file (rather than from input or fragment files) and other items imply that some input keys and some options to specific keys may be inaccessible when restart data are provided. In most cases supplying such inaccessible input options will simply be ignored; in some cases a warning is issued or an error abort occurs.

A restart file supplies data from a previous run that might be useful in the current one. The applications are (combinations are possible):

- Get a better start in the (first) SCF procedure by providing the electronic charge density (in the form of fit coefficients) from the preceding run,
- Continue an optimization by supplying the latest geometry (coordinates) from a previous run via the restart file (rather than typing them in),
- Get faster geometry convergence by supplying a Hessian,
- Breaking large jobs (Linear Transits, Frequencies) in smaller ones, each time doing a part and passing this on to the continuation run.

WARNING. The SCF and optimization procedures use *history* to improve convergence behavior. Most of such history information is not stored on a restart file. As a consequence, a restart may not continue exactly as the original run would have done if it hadn't terminated. In a SCF restart, for instance, the DIIS procedure has to rebuild the information. The same holds for geometry optimizations, although history plays usually not a very big role there.

The restart key

The name of the restart file must be provided with the key RESTART (see below). A list of data items is read from the file (if present on the file and only as far as significant for the new run) and used unless their usage is explicitly suppressed by the user.

```
RESTART restartfile { &
  optionlist
  optionlist
  ...
end}
```

`restart`

This *general* key can be used as a simple key - to supply the name of the restart file - or as a block key. In the latter case the continuation code (&) must be applied to tell the program that a data block follows.

`restartfile`

The name of a file with restart data. The path (absolute or relative) to the file must be included if the file is not local to the directory where the calculation executes. In most cases it will be a TAPE21 file from an adf calculation, but this is not necessary. It may be any file - constructed by the user for instance - provided it has the right structure. It must be a kf file and the data to be used must be stored in sections and under variable names as defined below, which is exactly how such data are generated by a normal adf run on TAPE21or on the checkpoint file TAPE13.
Note: the filename must not be one of the standard filenames used internally by the program, such as TAPE21, TAPE13 etc. Generally: don't use a name like tapenn where nn is a two-digit number.

`optionlist`

A list of options, separated by blanks or commas. The following options are applicable:

`noSCF`

Do not use any fit coefficients from the restart file as a first approximation to the (fitted) SCF density for the new calculation. Instead, the sum-of-fragments density will be used, as in a non-restart run.

`nogeo`

Do not use the geometry - Cartesian, Z-matrix, etc. coordinates - from the restart file.

`nohes`

Do not use any Hessian from the restart file.

Note: in the continuation of a Linear Transit, IRC or Frequencies run, geometric data are read from the restart file and will be used: the option nogeo is ignored. In a continued Frequencies run the input coordinates (key ATOMS) must be correct (i.e. the equilibrium geometry). In a continued LT or IRC run, the input coordinate values from atoms are ignored (but they must be supplied to give the program a preliminary count of atoms and fragments involved).

Structure of the restart file

All data that may be retrieved from the restart file must be stored in a specific location on the restart file. If you're simply using a TAPE21 result file or a TAPE13 checkpoint file you don't need to bother about this: adf has put all data in the right place; the following discussion is primarily for those who want to manipulate the restart file or even construct one themselves.

Since the restart file must be a kf file, the location of the data is of the form Section%Variable, specifying the section and the variable name. The section and variable names are case sensitive. See the utilities document for general information about kf files.

If the specified variable is not present in the specified section on the restart file - or if there is no such section at all - the data is not used, usually without an error message. In some cases a few global tests are carried out on the retrieved data; if they fail the tests the data are not used and a warning - in some cases an error abort - may be issued by the program.

KF files are binary files and so are the TAPE21 result file, the TAPE13 checkpoint file and generally any restart files. If you wish to edit and modify the contents, or just inspect them, the standard KF utilities can be used. Apply pkf to get a survey of the sections and variables on the file, dmpkf to get a complete ASCII version of the file and udmpkf to transform an ASCII version - presumably edited and modified - back into binary format. See the utilities document.

Data on the restart file

Follows a survey of all data items that the program may search for on the restart file.

SCF data

`Fit%coef_SCF`

The fit-expansion of the charge density to be used as start-up for the next SCF. Without these restart fit data the first SCF will start from the (fitted) sum-of-fragments charge density.

`Fit%coef_FreqCenter`

Only in a Frequencies run: the fit-expansion of the SCF-converged equilibrium geometry. It usually helps to get a somewhat better start-up of the SCF in displaced geometries.

If the noSCF option is used to the restart key, any Fit%coef_? data on the restart file are ignored.

Coordinates

`Geometry%xyz`

Cartesian atomic coordinates. The option nogeo suppresses using such data. In a Frequencies or continued Linear Transit run, they may be read but will be ignored (i.e. replaced by other coordinates data from the restart file, see below).

In most applications, when coordinates are read (and used) from the restart file, only *Cartesian* coordinates are retrieved and the corresponding Z-matrix values are computed from them, using the Z-matrix *structure* defined in the atoms data block. This is one of the reasons why the ATOMS key must be used even when the atomic coordinates are supplied on the restart file.

Hessian

```
GeoOpt%Hessian_cart
```

```
GeoOpt%Hessian inverted_cart
```

```
GeoOpt%Hessian_zmat
```

```
GeoOpt%Hessian inverted_zmat
```

All these four varieties are searched for if the new run searches for a restart Hessian matrix at all, that is: in an optimization, Linear Transit or Transition State search. As the names should suggest these variables stand for the Hessian, respectively the inverse of the Hessian in Cartesian or z-matrix coordinates.
In all cases the full square matrix must be present, with dimension the number of atomic coordinates, 3 times the number of atoms. This holds also for z-matrix coordinates. The 6 dummy coordinates play no role, the corresponding matrix elements in the Hessian should be zero.

If a Hessian is searched for on the restart file, all four possibilities above are tried and the first one found is used, the other ones being ignored. The order in which they are tried is:

If the current run uses *Cartesian* coordinates as optimization variables, then first the two cart varieties are tried, and vice versa for *z-matrix* optimization.

In a minimization (simple optimization or Linear Transit) first the *inverted* variety is tried; in a Transition State search the normal (not inverted) Hessian is looked for first.

Note**:** If a z-matrix Hessian is retrieved from the restart file the program will use the underlying z-matrix *structure* to derive a Cartesian Hessian from it. In such case the restart file must also contain:

```
GeoOpt%kmatrix
```

The z-matrix structure (references to the atoms in this matrix assume the ordering of atoms as used internally by the program).

Note: the kmatrix on the file need not be identical to the kmatrix used in the current calculation. In fact, the current calculation may not even have a z-matrix structure.

Transition State

In a continued TS run the program retrieves, apart from general geometry *optimization* data such as the Hessian - see above - only the latest TS search vector: the eigenvector of the (approximate) Hessian that points to the Transition State. All other TS-specific data are input-determined with corresponding defaults.

The ts search vector is stored in:

```
TS%mode to follow
```

A list of atomic coordinates (Cartesian or Z-matrix, depending on the type of optimization variables used. The underlying list of atoms has the atoms not necessarily in the order in which they have been given in input: rather they are grouped together by atom type.

Linear Transit

In a continued Linear Transit (lt) calculation the continuation run proceeds from where the previous run stopped. The total number of points by which the transit is scanned, the current point (its index and the Cartesian coordinates), the accumulated results of completed points on the transit etc. are copied from the restart file. If the restart file contains a section LT, then all relevant data must be present on it and correct (i.e. matching those of the current run: same number of LT *parameters*, and of course the same molecule.

`LT%nr of points`

The number of points by which the LT is scanned; this is identical to the Fortran variable ltimax in the code. The value on the restart file applies in the calculations and overwrites any input/default value (see the subkey *lineartransit* of the geometry block)

`lt%current point`

Index of the current LT scan point. This is where the program will continue. In a non-restart LT run, this index initializes at 1.

`LT%Energies`

An array with energy values, one for each LT point. When the LT run is completed, this array allows you to map out the energy along the LT path. The values for the completed LT points are stored on the restart file. This size of the array on the restart file must (at least) be the total number of points on the complete path.

`LT%Parameters`

Initial and final values for the LT parameters, which describe roughly the path (all other coordinates may be optimized at each point, depending on other input keys). The values from the restart file overwrite input values. The input values should be supplied, however, as if it were a non-restart run.

`LT%atmcrd`

zmat if a z-matrix structure is available for the molecule, cart otherwise. This is used to control printing of results. It does not define the type of optimization variables: see the next item.

`LT%geocrd`

zmat or cart: the type of optimization variables. This defines in which type of coordinates the LT parameters are defined and any optimization of other coordinates takes place.

`LT%xyz`

Cartesian coordinates for *all* LT points: 3*atoms*ltpoints. The size of the array must conform to this. Only the values of the completed LT points *and* those of the current point are relevant. Those of the current LT point are used as initial coordinates to start the current run.

`LT%zmatrix`

Same for the Z-matrix coordinates. They should match the Cartesian coordinates for the completed LT points (this is not checked). Those for the current LT point will be recomputed from the current *Cartesian* coordinates.

IRC

In a continued Intrinsic Reaction Coordinate (IRC) calculation, the continuation run processes the path(s) as specified in input. Any info for such path(s) on the restart file will then be used to continue from there. If the restart file contains the relevant IRC sections, see below, then all relevant data must be present on it and correct (i.e. matching those of the current run.

The sections on file pertaining to the IRC are:

IRC: this section contains information about the central (TS) point, which variables are optimized in each of the IRC points, the connection matrix defining the z-matrix structure, etc.

IRC_Forward and IRC_Backward: these sections contain the data of the two paths from the Transition State down to the two adjacent local energy minima: for each point the distance from the previous point and the local curvature and molecular properties such as energy, atomic charges and dipole moment.

`LT%nr of points`

The number of points by which the LT is scanned; this is identical to the Fortran variable ltimax in the code. The value on the restart file applies in the calculations and overwrites any input/default value (see the subkey *lineartransit* of the geometry block)

`LT%current point`

Index of the current LT scan point. This is where the program will continue. In a non-restart LT run, this index initializes at 1.

`lt%Energies`

An array with energy values, one for each LT point. When the LT run is completed, this array allows you to map out the energy along the LT path. The values for the completed LT points are stored on the restart file. This size of the array on the restart file must (at least) be the total nr of points on the complete path.

`lt%Parameters`

Initial and final values for the LT parameters, which describe roughly the path (all other coordinates may be optimized at each point, depending on other input keys). The values from the restart file overwrite input values. The input values should be supplied, however, as if it were a non-restart run.

`lt%atmcrd`

zmat if a z-matrix structure is available for the molecule, cart otherwise. This is used to control printing of results. It does not define the type of optimization variables: see the next item.

`lt%geocrd`

zmat or cart: the type of optimization variables. This defines in which type of coordinates the LT parameters are defined and any optimization of other coordinates takes place.

`lt%xyz`

Cartesian coordinates for *all* LT points: 3*atoms*ltpoints. The size of the array must conform to this. Only the values of the completed LT points *and* those of the current point are relevant. Those of the current LT point are used as initial coordinates to start the current run.

`LT%zmatrix`

Same for the Z-matrix coordinates. They should match the Cartesian coordinates for the completed LT points (this is not checked). Those for the current LT point will be recomputed from the current *Cartesian*

coordinates.

Frequencies

In the continuation of a Frequencies calculation all Frequencies-related data are retrieved from the section Freq on the restart file. (SCF fit data are, as always, retrieved from the section Fit). A fairly large number of items will be read and must all be present (*if* a section Freq is present in a restart file supplied to a Frequencies run). Technical parameters such as the type of numerical differentiation, size of displacements etc. are read from the restart file. Any input specifications are ignored.

`Freq%kountf`

Counter of number of geometries completed. In a non-restart run this is initialized at zero; in a restart it is read from the file.

`Freq%nraman`

Flag for RAMAN calculations

`Freq%numdif`

1 or 2: defines numerical differentiation used to compute the force constants from the gradients in slightly displaced geometries (by 1-point or 2-point differentiation).

`Freq%disrad`

Size of displacement for cartesian or bond-length displacements.

`Freq%disang`

Size of angular (bond angle, dihedral angle) displacements.

`Freq%atmcrd`

zmat or cart: specifies whether a z-matrix structure is present. This does not define the type of displacement coordinates, see the next item.

`Freq%geocrd`

Type of coordinates in which the displacements are carried out: zmat or cart

`Freq%nfree`

Number of free and independent displacement variables.

`Freq%idfree`

References from the atomic coordinates (in internal order) to the independent displacement variables.

`Freq%all freedoms`

(logical) flags whether or not the complete energy surface is scanned around the equilibrium or only part of the internal degrees of freedom are used.

`Freq%xyz`

equilibrium coordinates (internal order of atoms).

`Freq%kmatrix`

Z-matrix structure. Pointers are indexed by and refer to atoms in the internally used order.

`Freq%zmatrix`

Z-matrix coordinates of the equilibrium geometry (internal ordering of atoms).

`Freq%rigids`

6 rigid motion vectors (one may be zero, in case of a linear molecule). Each vector has as many components as there are atomic coordinates. The values correspond to the internal ordering of atoms.

`Freq%xyz displaced`

Cartesian coordinates of displaced geometry to carry out now. In a non-restart run this would be the equilibrium geometry.

`Freq%zmatrix displaced`

Similar for the Z-matrix coordinates.

`Freq%Dipole previous`

dipole vector (3 components) for the last geometry handled.

`Freq%Dipole`

dipole at the equilibrium geometry.

`Freq%Dipole derivatives`

Derivatives of the dipole wrt atomic coordinate displacements.

`Freq%Gradients previous`

Energy gradients (derivatives wrt atomic coordinate displacements) in the last handled geometry.

`Freq%Force constants`

Matrix of force constants. This is, together with the *Dipole derivatives* the final quantity to compute. At each cycle of the Frequencies data are added to it. Upon completion of the Frequencies cycles the frequencies and normal modes are computed from it. Together with the dipole derivatives it then also yields the InfraRed intensities.

Printed Output

The amount of printed output is regulated with the keys Print, NoPrint, EPrint and Debug. (No)Print and Debug are simple keys, EPrint is a block type key.

Many print options pertain to debugging situations and are included here only for completeness. This section is intended to give a survey of all possibilities. Some items may be mentioned again in other sections where the subject of a particular print switch is discussed.

Print / NoPrint

```
PRINT Argumentlist
Print Argumentlist
NoPrint Argumentlist
```

`Argumentlist`

A sequence of names separated by blanks or commas.
The keys Print and NoPrint may occur any number of times in the input file. The names in the argument list may refer to various items. For some of them printing is normally on, and you can turn them off with NoPrint. For others the default is not printing; use Print to override that.

Follows a list of the recognized items that are applicable in the argument lists, with a short explanation and defaults. Item names must be used exactly as given in the table - abbreviated or elongated forms will not be recognized - but they are not case sensitive.

Item	*Default*	*Explanation*
$ALL	No	Turns on *all* print options. This will not be affected by any additional noprint instructions. Be careful: this generates a large amount of output. To be used only for debugging purposes.
Atdist	No	Inter-atomic distance matrix at each new geometry (in an optimization)
Bas	Yes	General control of output related to elementary basis functions (bas).
BlockCheck	No	Intermediate data during the determination of the block length. (see Blocks)
Blocks	No	Numerical integrals, consisting of loops over large numbers of points, are split up in loops over blocks of points. The block length is determined by the available amount of workspace. Given this amount, the maximum block lengths, according to memory usage in a few relevant routines, are computed (and printed with this print option) and used to impose upper bounds on the block length actually use.
Character-Table	No	Table of characters for the irreducible representations of the pointgroup symmetry.
Computation	Yes	Reports progress of the computation, with (concise) info about each SCF cycle and each Geometry update in an optimization.
Core	No	Description of the frozen core: frozen core expansion functions (corbas) and the expansion coefficients for the frozen orbitals. This printing can only be activated if Functions is also *on*, otherwise it is ignored.
CoreOrt	No	The valence basis set contains auxiliary Core Functions. They are not degrees of freedom but are used solely to ensure orthogonalization of the valence set to the frozen Core Orbitals. The orthogonalization coefficients and some related overlap matrices are printed.
CoreTable	No	Internally the charge density and potential of the atomic frozen cores are processed as tables with values for a sequence of radial distances. A few initial and a few final values from these tables are printed, along with the (radial) integral of the core density, which should yield the number of core electrons.
EKin	No	At the end of SCF: Kinetic energy of each occupied MO.
EndOf	No	Flags the exit from a few major routines, with cpu times used in these modules. Primarily a debug tool.
EPauli	Yes	The repulsive Pauli term in the bonding energy (also called exchange repulsion) with its decomposition in density functional (lda and nl) and Coulomb terms.
Fit	Yes	General control of output related to the density fitting.
Fmat	No	Fock matrix computed at each cycle of the SCF.
FmatSFO	No	Fock matrix in the basis of symmetrized fragment orbitals (SFOs). This option requires the FULLFOCK and ALLPOINTS keyword to be present

		in the input. The matrix is printed only at the first and the last SCF cycle.
ForceConstants	Yes	Force constants matrix (Frequencies run only)
FreqHess	No	matrix of force constants (Frequencies run) after each applicable step in its processing: transformation from/to Cartesian and Z-matrix coordinates, symmetrizations, ...
Frag	No	General control of output related to build-molecule-from-fragments.
Functions	Yes	List of employed Slater-type exponential basis functions and fit functions.
Gradients	No	detailed info of computed energy gradients (in optimization runs)
Group-Operators	No	3*3 matrices of pointgroup symmetry operators, with the axis and angle of rotation
HessEig	No	Eigenvalues of the Hessian in each cycle of a Geometry Optimization. The print-out in the intermediate cycles is suppressed if output of updated coordinates etc. is turned off (see the eprint subkey Repeat (option GeoStep).
Idfree	No	List of free atomic coordinates with indication whether they are optimization coordinates (this info is also contained in the output of new atomic coordinates at each step of an optimization)
Inertia	No	Warning message in the log file in case of zero product of moments of inertia (this may correctly be the case for certain molecules)
Inputkeys	No	List of keys that were specified in input, together with some of the associated data. The list is printed directly after the echo of the Input File, before the header with adf program information. A few special keys will not be echoed: (No)Print,(No)Skip, Allow.
Irrep-Matrices	No	Irreducible representation matrices
Logfile	Yes	At the end of the calculation a copy of the log file is appended to standard output
low	No	Construction of the LOW basis from the elementary BAS functions and from the SFOs: combination coefficients
lowMO	No	MOs are printed in the LOW (Lowdin) representation, in the RESULTS section
OvlBAS	No	Overlap matrices processed during the construction of the LOW basis
Parser	No	Most input records are echoed twice (at the very beginning of output). First the original version, then the parsed version in which expressions have been replaced, redundant blanks removed, etc. The parsed version is what the program really uses as input. Comment blocks, and function definitions (in define blocks) are not parsed, and are not affected by this switch. If the print switch is off only the original, non-parsed input record is echoed in output. This print switch affects only the part of input after its occurrence.
Pmat	No	The density matrix (in Lowdin representation) in each cycle of the SCF.
QMpot	Yes	At the end of the SCF for each atom the electrostatic potential at its nucleus (excluding its own contribution of course).
	No	Redundant Coordinates used in the construction of the initial - force field derived - Hessian
RedCrdBonds	No	atom-atom bonds determined for the construction of the initial Hessian
RedCrdH	No	Hessian in the redundant coordinates representation
SCF	Yes	Controls the information about progress of the SCF procedure. Applies only if the print switch computation is on.
sdiis	No	Expansion coefficients applied by the DIIS procedure during the SCF.

sdiismat	No	Turns on sdiis(see above) *and* prints the *error vector* constructed by the DIIS routine (this is the commutator of the Fock matrix and the Density matrix). This is used to determine the DIIS expansion coefficients and to assess convergence.
SFO	Yes	General control of SFO-related output. If turned off, (almost) all such output is suppressed. If on, as is the case by default, such printing is controlled by the eprint subkey SFO.
Smat	No	Overlap matrix of BAS functions.
Smearq	No	Smear parameter - if and when applied - used in the determination of electronic occupation numbers for the MOs, with details of how it works out at every cycle of the SCF. For debugging purposes.
SpinOrbit	No	detailed information about how double-group symmetry representations are related to the single group representations
Tails	No	In each block of integration points (see Blocks) the evaluation of (Slater-type) exponential functions (basis, fit) is skipped when the function has become negligible for all points in that block due to the distance of those points from the atom where the function is centered. The relative savings due to this distance screening is printed at the first geometry cycle (use debug for printing at all cycles).
TechPar	Yes	Technical parameters such as maximum vector length in vectorized numerical integration loops, SCF and Geometry Optimization strategy parameters.
Timing	No	Print out of more timing info (in particular with respect to performance of the parallel version of adf) than is provided by the standard Timing Statistics tables at the end of each output.
TimingDetail	No	Similar, but more details.
TimingTooMuchDetail	No	Similar, but even worse.
Workspace	No	Statistics of calls to the Workspace Manager (memory management).
Arguments for the keys print and noprint.		

For print switches that start with Frag.., Fit.., Freq..,Geostep..., Numint.., Repeat..., SCF..., TF..., see the key EPRINT below.

Debug

The key DEBUG is used to generate extensive output that is usually only relevant for debugging purposes. It operates exactly like the PRINT key but there is no converse: nodebug is not recognized; it would be irrelevant anyway because by default all debug print switches are off.

A list of the possible items for the DEBUG key is given below.

All items of the print list can also be used with the debug key. If they are not mentioned in table III, the meaning is the same as for the print key, but the corresponding output may be generated more often, for instance at every SCF cycle rather than at the last one only.

Item	*Explanation*
Basis	Construction of the orthonormal LOW basis from elementary (BAS) and fragment (FO) basis.
Core	Core Orthogonalization procedure
Ekin	Kinetic energy matrices. (compare the print switches EKIN)
Fit	Construction of the symmetry adapted fit functions

Fitint	Construction of integrals used in the Fit procedure.
FmatSFO	Requests printing of the SFO overlap matrix in addition to the Fock matrix. This overlap matrix must be identical to the one printed in the results section of the output. This way one makes sure that the AO to SFO transformation of the Fock matrix has been carried out properly.
Freq	Force matrices processed in the computation of frequencies: Cartesian and internal representation, before and after symmetrization, etc. (as far as applicable).
GeoStep	Geometry optimization procedure. All relevant items.
Hess	Complete eigensystem of the Hessian during geometry optimizations.
NumInt	Numerical integration. Very extensive output (including the coordinates and weights of all generated points).
Pmat	P-matrix (density matrix) during SCF and in the ETS analysis program in the BAS representation.
Rhofih	Computation of fit coefficients during the SCF.
SCF	Extensive output during the SCF procedure about many different items. See also EPRINT, subkey SCG.
SDIIS	All data concerning the DIIS as used during the SCF. See ERPRINT, subkey SDIIS.
TransitionField	The Transition State procedure to compute and analyze certain terms in the bonding energy. The distinct components, the involved transition field Fock matrices, etc.

Table III. Arguments for the print key DEBUG. All debug switches are by default off.

Eprint

The key EPRINT is an extended version of the (no)print key, employed for print switches that require more specification than just off or on.

Contrary to what is the case for the keys print and noprint, the key EPRINT must occur only once in the input file; any subsequent occurrences are incorrect and ignored or lead to abort.

```
EPRINT
   subkey
   subkey
   ...
end
```

`subkey`

A subkey-type structure: it consists of a keyword followed by data, so that it functions as a simple (sub)key, *or* it is a keyword followed by a data *block* which must then end with the word subend.

The subkeys used in the eprint data block are called Eprint keys. A complete list of them is given below. All available eprint keys are discussed in the schemes below. The enclosing records eprint and end are omitted in these schemes.

EPRINT subkeys	*Subject*
AtomPop	Mulliken population analysis on a per-atom basis
BASPop	Mulliken population analysis on a per-bas-function basis
Eigval	One-electron orbital energies
Fit	Fit functions and fit coefficients

Frag	Building of the molecule from fragments.
FragPop	Mulliken population analysis on a per fragment basis
Freq	Intermediate results in the computation of frequencies (see debug: freq).
GeoStep	Geometry updates (Optimization, Transition State, ...)
NumInt	Numerical Integration
OrbPop	(Mulliken type) population analysis for individual MOs, both on a per-SFO basis and on a per-bas function basis. In a SpinOrbit calculation no SFO-type analysis is available (not yet implemented).
OrbPopER	Energy Range (ER) in hartree units for the OrbPop subkey
Repeat	repetition of output in Geometry iterations (SCF, optimization, ...)
SCF	Self Consistent Field procedure
SFO	Information related to the Symmetrized Fragment Orbitals and the analysis (populations and MO coefficients) in this representation.
TF	Transition Field method for the evaluation and analysis of certain bonding energy terms.

Table IV. List of eprint subkeys.

Eprint subkeys vs. Print switches

Several eprint subkeys are merely shortcuts for normal (no)print switches. All such simple subkeys are used in the following way:

```
ESUBKEY argumentlist
```

`Esubkey`

One of the following eprint subkeys: Fit, Frag, GeoStep, NumInt, Repeat, SCF, sdiis, SFO, TF, Time.

`argumentlist`

A sequence of names, separated by delimiters. Each of these names will be concatenated with the esubkey and the combination will be stored as a normal print switch.
Example:
Frag rot, SFO
will be concatenated to fragrot and fragsfo and both will be stored as print switches. All such combinations can also be specified directly with the key PRINT. The example is therefore exactly equivalent with the input specification:
print FragRot, Fragsfo

If any of the names starts with the two characters no, the *remainder* of the name will be concatenated with the eprint, but now the result will be stored and treated as a noprint switch. Items that are on by default can in this way be turned off. Example:

```
EPRINT
 FRAG noRot Eig
end
```

This turns Rot *off* and Eig *on* for the eprint subkey Frag. Equivalent would be:

```
NOPRINT FragRot
Print FragEig
```

Follows a description of all simple EPrint subkeys:

Fit

The subkey fit controls output of how the elementary fit functions are combined into the symmetric (A1) fit functions. It controls also printing of the initial (start-up) and the final (SCF) fit coefficients.

```
FIT list
```

`list`

A list of items, separated by blanks or commas. The following items are recognized: Charge, Coef, Comb.

`Charge`

The amount of electronic charge contained in the fit (start-up), total and per fragment.

`Coef`

The fit coefficients that give the expansion of the charge density in the elementary fit functions.

`Comb`

The construction of the totally symmetric (A1) fit function combinations from the elementary fit functions.

By default all options are off.

Frag

The subkey frag controls output of how the molecule is built up from its fragments.

```
FRAG list
```

`list`

A list of items, separated by blanks or commas. The following items are recognized: Eig, Fit, Rot, SFO.

`Eig`

The expansion coefficients in elementary functions (bas) of the fragment Molecular Orbitals as they are on the fragment file.

`Rot`

The rotation (and translation) required to map the master fragment (i.e. the geometrical data on the fragment file) onto the actual fragment which is part of the current molecule.
N.B.: if eig and rot are both *on*, the rotated fragment orbitals are printed also.

`Fit`

The fit coefficients that describe the fitted charge density of the fragments after the rotation from the *master* fragment on file to the actual fragment. These are the molecular fit coefficients that are used (by default) to construct the total molecular start-up (fitted) charge density and hence the initial Coulomb and XC potential derived from it.

`SFO`

The Symmetry-adapted combinations of Fragment Orbitals that are used in the current calculation. This feature ensures that the definition of the SFOs is printed. This will happen anyway whenever the eprint subkey SFO itself is activated.

By default all options are off.
Remark: SFO analysis in a Spin-Orbit relativistic calculation is implemented only in the case there is one scalar relativistic fragment., which is the whole molecule.

Freq

Controls printing of Force matrices and a few more data that are intermediate results in the computation of frequencies after all coordinate displacements have been carried out.

```
FREQ list
```

`list`

contains any of the items SymCoord, DMuRot, Hess.

`SymCoord`

print the Symmetry Coordinates, both as they are generated, and some related info in their processing later on. The Symmetry Coordinates are symmetry-adapted combinations of cartesian displacements, with the pure translations and rotations projected out.

`dmuRot`

info about generating Dipole-Derivative information in transformation between cartesian and internal coordinate representation as regards the rotational aspects.

`Hess`

processing of the completed matrix of force constants, symmetrization, transformation to other coordinates.

By default all options are off.

GeoStep

Controls output concerning the geometry update method, parameters, energy gradients, etc. It plays no role in a SinglePoint calculation.

```
GEOSTEP list
```

`list`

A list of items, separated by blanks or commas. The following items are recognized: Energy, GradientTerms, Gradients, Upd.

`Energy`

summary of the (bonding) energy and its components as computed in the geometry update procedure.

`Gradients`

Energy gradients on the free variables. These may be all or some of the cartesian or the Z-matrix coordinates, depending on the case.

`GradientTerms`

The decomposition of the gradients in computed terms, as described in the thesis of L.Versluis [7].

`Upd`

parameters used and adapted in the geometry update procedure.

By default Gradients, Upd are on, the other items off.

NumInt

Output related to the numerical integration procedure: parameters, generated points, tests on the accuracy of the generated scheme, etc.

```
NUMINT list
```

`list`

A list of items, separated by blanks or commas. The following items are recognized: All, Geo, Ovl, Par, Pnt, Res, Sym, Test.

`All`

includes all other options and prints in addition the coordinates and weights of all generated points. *This can be a lot of output!*

`Geo`

geometric data such as boundary planes around the molecule, as they are computed and used in the program section where the point grid is generated.

`Ovl`

numerically integrated are the auto-overlaps of symmetry-adapted combinations of elementary basis functions SBAS. The deviations from the analytically computed values is printed. The test option, see below, yields a summary of these data: the maximum error and the root-mean-square error.

`Par`

employed precision parameters, atomic spheres radii etc.

`Pnt`

the generated *numbers* of points in each of the subregions processed in the point-generating procedure.

`Res`

results as regards the total number of points, the sum-of-weights and the partitioning of the points in blocks (for segmented vectorization).

`Sym`

the symmetry operators that are computed directly from the coordinates (irrespective of the input Schönfliess symbol) and that are used to construct thec numerical integration grid in a symmetric fashion.

`Test`

a few external tests are performed after the grid has been generated, such as the numerical integration of the sum-of-fragment densities. See also the norms option.

By default Res and Test are on, the other options off.

OrbPop

Specifies that (Mulliken type) population analysis should be printed for individual MOs, both on a per-SFO basis and on a per-bas function basis. The format of the subkey is as follows:

```
ORBPOP TOL=X Nocc Nunocc
SUBEND
```

X is the threshold for the SFO coefficient value to include in the listing for the per-SFO analysis. Nocc is the number of the highest occupied and Nunocc is the number of the lowest unoccupied orbitals to analyse.

OrbPopER

Specifies the energy range for the MOs to which the OrbPop key applies. The default range is from -0.7 below the HOMO to 0.2 hartree above the LUMO. Usage:

```
OrbPopER minEn maxEn
```

where minEn and maxEn are both in hartree, and have the defaults just specified. In order to get information on many more orbitals, simply specify a large negative value for minen and a large positive value to maxen.

Repeat

Control the repetition of output in Geometry iterations: optimization, computation of frequencies, transition state search.

```
Repeat list
```

`list`

contains one or more of the following items: NumInt, SCF.

`NumInt`

Output from the numerical integration procedure, like parameters, numbers of points generated, test data is controlled by the *numint* subkey (see below). The *repeat* subkey controls whether the output is repeated for all geometries (if the flag is on) or only for the first (if the flag is off). Some concise info is produced (repeatedly) anyway if the print switch computation is on.

`SCF`

Controls similarly the SCF output, like population analysis and orbital eigenvalues. If the flag is on, these items are printed at the last SCF cycle in every geometry, otherwise only at the last (in case of an optimization, not in case of a Frequencies calculation).

By default both options are off.

SCF

Output during the SCF procedure.

```
SCF list
```

`list`

is a list of items, separated by blanks or commas. The following items are recognized: Eigval, Eigvec, Err, Fmat, Keeporb, MOPop, Occ, Pmat, Pop, Start.

`Eigval`

Eigenvalues of the one-electron orbitals at the last SCF cycle. In a run with multiple SCF runs (Geometry Optimization,..) this printing occurs only for the last SCF procedure. See also the eigval subkey of EPRINT. (Use debug or the *repeat* subkey of EPRINT to get output on *all* cycles).

`Eigvec`

MO eigenvector coefficients in the BAS representation. Only printed on the last SCF cycle.

`Err`

SCF error data which are checked for convergence. By default this takes effect after cycle 25 of the SCF. If the key is set it takes effect at the first cycle. Optionally one may type ErrN,where n is an integer (written directly after Err without a blank in between), in which case the key takes effect at cycle n.

`Fmat`

Fock matrix in the low representation.

`Keeporb`

If the KeepOrbitals option is activated (see the key SCF), output is generated whenever this option actually results in a change of occupation numbers as regards the energy ordering.

`Occ`

concise output of SCF occupation numbers on last SCF cycle if no eigenvalues are printed (see: Eigval).

`moPop`

Mulliken populations in terms of the elementary basis functions (bas), per MO, for input-specified MOs (see the eprint subkey *orpop*)

`Pmat`

Density matrix

`Pop`

General control of bas Mulliken populations. This supervises all printing (whether populations are printed or not) according to the eprint subkeys *atompop, fragpop, orbpop* (the latter only as regards the bas population analysis at the end of the SCF procedure).

`Start`

Data pertaining to the *first* SCF cycle (of the *first* SCF procedure, in case of an optimization; use *repeat* to get this for *all* SCFs).

By default Eigval, Keeporb, Occ, and Pop are on, the others off.

SFO

Information pertaining to the use of Symmetrized Fragment Orbitals (for analysis purposes).

```
SFO list
```

`list`

A list of items, separated by blanks or commas. The following items are recognized: eig, eigcf, orbpop, grosspop, fragpop, ovl.

`Eig`

The MO coefficients in terms of the SFOs.

`Eigcf`

idem, but now also containing the coefficients pertaining to the CoreFunctions.

`OrbPop`

population analysis of individual orbitals. The orbitals analyzed are set with the eprint subkey *orbpop*.

`GrossPop`

Gross populations of the SFOs, split out in symmetry representations. GrossPop is automatically turned on when OrbPop is activated.

`FragPop`

Population analysis on a per-FragmentType basis. This analysis does in fact not depend on the SFOs (ie, the result does not depend on how the SFOs are defined), but the computation of these populations takes place in the SFO-analysis module, which is why it is controlled by the SFO print option. FragPop output is given per orbital when OrbPop is activated, per symmetry representation when GrossPop is activated, and as a sum-over-all-orbitals-in-all-irreps otherwise (if FragPop is active).

`Ovl`

Overlap matrix of the SFO basis, separately for each symmetry representation.

By default eig,orbpop,ovl are on, the other options off.

In a Spin-Orbit calculation the SFO analysis is not yet implemented completely.

Remark: the options eig, eigcf replace the previous (now disabled) simple print options eigsfo and eigsfo.

Note that the simple print key SFO controls whether or not the eprint subkey *sfo* is effective at all.

TransitionField

Part of the bonding energy is computed and analyzed by the so-called Transition State procedure [3, 110]. This has nothing to do with physical transition states, but is related to the Fock operator defined by an average charge density, where the average is taken of the initial (sum-of-orthogonalized-fragments) and the final (SCF) charge density. There is also an analogous term where the average is taken of the sum-of-fragments and the sum-of-orthogonalized-fragments. Various terms, Fock operators and Density Matrices used in this approach may be printed. To avoid confusion with real Transition States (saddle points in the molecular Energy surface) the phrase TransitionField is used here.

```
TF list
```

`List`

A list of items, separated by blanks or commas. The following items are recognized: Energy, Fmat,

DiagFmat, FragPmat, DiagFragPmat, F*dPmat, DiagF*dPmat, OrbE.

`Energy`

Energy terms computed from the TransitionField.

`Fmat`

TransitionField Fock matrices.

`DiagFmat`

Idem, but only the diagonal elements.

`FragPmat`

The molecular P-matrix constructed from the sum-of-fragments.

`DiagFragPmat`

idem, but only the diagonal elements.

`F*dPmat`

The TransitionField energy term can be expressed as a Fock operator times the difference between two P-matrices (initial and final density).

`DiagF*dPmat`

only diagonal elements

`OrbE`

Orbital energies in the TransitionField.

By default all options are off.

Other Eprint subkeys

We discuss now the remaining eprint sub keys that are not simple shortcuts for print switches.

Orbital Energies

```
Eigval noccup {nvirtual}
```

This specifies the *number* of one-electron orbitals for which in the SCF procedure energies and occupation numbers are printed whenever such data is output: the highest noccup occupied orbitals and the lowest nvirtual empty orbitals. Default values are noccup=10, nvirtual=10. If only one integer is specified it is taken as the noccup value and nvirtual is assumed to retain its standard value (10). Printing can be turned off completely with the eprint sub key SCF, see above.

Mulliken Population Analysis

All population subkeys of eprint refer to *Mulliken* type populations.

```
ATOMPOP level
```

Populations accumulated per atom.

level must be none, gross or matrix. none completely suppresses printing of the populations; gross yields the gross populations; matrix produces the complete matrix of net and overlap populations. Default value: matrix.

```
BASPop level
```

Populations are printed per elementary (bas) basis function. The level options are none, short, gross, matrix. none, gross and matrix are as for atompop.

short yields a summary of BAS gross populations accumulated per angular momentum (*l*) value and per atom.

Default value: gross.

```
FragPop level
```

Completely similar to the atompop case, but now the populations per *fragment*. Of course in the case of single-atom fragments this is the same as atompop and only one of them is printed. Default: matrix.

For all three population keys atompop, fragpop and baspop, specification of a higher level implies that the lower-level data, which are in general summaries of the more detailed higher level options, are also printed.

Printing of any populations at the end of the SCF procedure is controlled with the eprint sub key *SCF* (pop).

Population Analysis per MO

A very detailed population analysis tool is available: the populations *per orbital* (MO). The printed values are independent of the occupation numbers of the MOs, so they are not populations in a strict sense. The actual populations are obtained by multiplying the results with the orbital occupations.

The analysis is given in terms of the SFOs and provides a very useful characterization of the MOs at the end of the calculation, after any geometry optimization has finished. This feature is now also available in a Spin-Orbit coupled relativistic calculation, in the case there is one scalar relativistic fragment, which is the whole molecule.

The same analysis is optionally (see EPRINT subkey *SCF*, option mopop also provided in terms of the elementary basis functions (bas).

```
OrbPop {noccup {nvirtual}} {tol=tol}
 subspecies orbitals
 subspecies orbitals
 ...
subend
```

noccup

Determines how many of the highest occupied orbitals are analyzed in each irrep. Default noccup=10.

nvirtual

Determines in similar fashion how many of the lowest virtual orbitals are analyzed in each irrep. Default nvirtual=4.

tol

Tolerance parameter. Output of SFO contributions smaller than this tolerance may be suppressed. Default: 1e-2.

`subspecies`

One of the subspecies of the molecular symmetry group. Can not be used (yet) in a Spin-Orbit coupled calculation.

`orbitals`

A list of integers denoting the valence orbitals (in energy ordering) in this subspecies that you want to analyze. This overrules the noccup,nvirtual specification for that symmetry representation. In an unrestricted calculation two sequences of integers must be supplied, separated by a double slash (//).

Any subset of the subspecies can be specified; it is not necessary to use all of them. No subspecies must occur more than once in the data block. This can not be used in a Spin-Orbit coupled equation (yet).

A total SFO gross populations analysis (from a summation over the occupied MOs) and an SFO population analysis per fragment type are preformed unless *all* MO SFO-populations are suppressed.

Mayer Bond orders

The Mayer bond orders are calculated and printed if the keyword

```
EXTENDEDPOPAN
```

is included in the input. Next to the Mayer bond orders Mulliken atom-atom populations per l-value will be calculated and printed if this keyword is included in the input. Note that this keyword is not a subkey.

Reduction of output

One of the strong points of adf is the analysis in terms of fragments and fragment orbitals (SFOs) that the program provides. This aspect causes a lot of output to be produced, in particular as regards information that pertains to the SFOs.

Furthermore, during the SCF and, if applicable, geometry optimizations, quite a bit of output is produced that has relevance merely to check progress of the computation and to understand the causes for failure when such might happen.

If you dislike the standard amount of output you may benefit from the following suggestions:

If you are not interested in info about progress of the computation:

```
NOPRINT Computation
```

If you'd like to suppress only the SCF-related part of the computational report and make the GeometryUpdates related part more concise:

```
NOPRINT SCF, GEO
```

(Keep computation on, so you get at least some info about the GeometryUpdates)

If you don't want to see any SFO stuff:

```
NOPRINT SFO
```

To keep the SFO *definitions* (in an early part of output) but suppress the SFO-mo coefficients and the SFO overlap matrix:

```
EPRINT
 SFO noeig, noovl
END
```

Note: the SFO-overlap matrix is relevant only when you have the SFO-MO coefficients: the overlap info is needed then to interpret the bonding/anti-bonding nature of the various SFO components in an MO.

If you are not interested in the SFO *populations*:

```
EPRINT
 SFO noorbpop
END
```

ASCII Output Files with Atomic Coordinates

You may want to have a special result file that contains the atomic coordinates corresponding to all the geometries processed in the calculation, for instance to feed it to a 'movie' generator to display the development of an optimization run. This is regulated with the key FILE:

```
FILE filetype filename { filetype2 filename2 }
```

`filetype`

Specifies the format of the output. Currently supported are three varieties: MOPAC, mol and xyz

`filename`

The file to which the output is written; the file should not yet exist. The name may include a full or relative path with respect to the directory where the calculation runs.

The same input record may contain any number of pairs-of-arguments, for instance to specify that both a mol-type *and* a xyz-type result file are to be generated. The key may also occur more than once in the input stream, in which case the argument lists are effectively all concatenated (by the program).

2.3 More Options

We continue with keys and other aspects of input that are less important in most applications. A few keys have been mentioned already before but allow additional or alternative usage to be discussed now.

Some keys may easily be misused yielding ridiculous results, possibly without any warning or relevant message to this effect from the program. This applies in particular, but not exclusively, to the sections Precision and Control of Program Flow

General

Link-in Input files

Part of the input file can be put into a separate ASCII file, which can be addressed from the (standard) input stream:

```
INLINE inlinefile
```

inlinefile must be the name of the auxiliary ASCII file (including its path, absolute or relative to the run-directory). When inline is encountered in the input file, adf opens the specified file and continues reading from that file as if it were in-line expanded into the input file. When the end-of-file is encountered reading resumes from the original file.

The contents of the inlinefile must *not* end with end input, unless you wish to terminate all input reading at that point.

InLine may occur any number of times in the input file. Use of inline may also be nested (up to 10 levels): the key INLINE may be used in the inlinefile in the same fashion as in the standard input file.

The inline feature makes it easy to pack your preferred settings that are not matched by the program's defaults in one file and use them in every run with a minimum of input-typing effort. Obvious applications are output control (print) settings and precision parameters.

Note: you can *not* use inline to store parallel settings, not even by using inline on the first line of your input and placing the parallel keyword on the first line of the inlinefile: before opening the inlinefile and expanding it into the inputfile, the program has already detected that the first line of input does not specify the parallel settings.

Title and Comment

```
TITLE Title
```

Title may be any string. The program combines it (that is, the first approximately 50 characters) with date and time of the job to construct the *job identification.* The job identification is used to stamp an identification on result files, which will be read and printed if such a file is used again, for instance as a fragment file.

The job identification will also be echoed in the output header to identify the current run. By default the date and time are combined with a dummy string. In Create mode the title is first read from the data file that supplies the basis functions etc and can then be overwritten via input.

Note that, contrary to some other programs, adf does *not* take the first input record as a title. Typing your title as the first record, *without starting the record with the keyword* title, may produce very strange results: ADF will try to interpret the first word on that line as a keyword, possibly abbreviated!

You can put more remarks in the input file to be echoed in the standard output file; these will not become part of the job identification:

```
COMMENT
  text
  ...
end
```

The text records are copied to the output header, directly after the job identification. Expressions are not parsed and constants or functions are not replaced: it is a straightforward copy.

The key COMMENT may occur any number of times; all text blocks are printed in the output header with a blank line between any two text blocks.

Layout of input

Empty records and leading blanks in records are allowed and ignored, and can be used to enhance clarity and readability of the input file for human readers.

An exclamation mark (!) is interpreted by the input reading routine as denoting the end-of-line. Instead of the exclamation mark you may also use a double colon (::). The part of the line after the exclamation mark (double colon) - including the ! or :: itself - is ignored. In this way one can include comments and clarifying remarks, which will not be echoed in the output header (compare the key COMMENT).

Geometry

Orientation of Local Atomic Coordinates

As discussed before the atomic positions are input with the key ATOMS. One option has thus far not been mentioned: the possiblity to redefine the local coordinate frame of an atom.

```
ATOMS {type of coordinates}
  {n} atomname coordinates {F=fragment} {Z=xx yy zz}
  ...
end
```

Except for the z= option all aspects have been examined already before.

`z=xx yy zz`

defines a reorientation of the local atomic z-axis; it is interpreted as a direction vector with components (xx,yy,zz) pointing away from the atom. In the local, reoriented frame the local atomic x-axis will be rotated to the plane defined by the directions of the molecular z-axis and the local atomic z-axis.
This feature can be used only for single-atom fragments (otherwise it is ignored). Its purpose is to give more flexibility in the analysis of the final molecular orbitals in terms of the atomic orbitals. In such a case it may be very helpful to redefine the orientation of say the p-orbitals of an atom. For instance, you may orient all p-orbitals towards the origin by specifying for each atom z= -x -y -z (with x,y,z the coordinates of that atom). By default the local and molecular z-axes are identical.

Symmetry

Together with the point group symmetry , a tolerance parameter can be supplied.

```
SYMMETRY {symbol} {tol=tolerance}
```

`symbol`

The Schönfliess symmetry symbol. A complete list of allowed values for this argument is given in Appendix 3.

`tolerance`

The tolerance (absolute deviation in the Cartesian coordinates) for atomic positions being symmetry equivalent. The same tolerance applies to check the mapping of fragments on attached fragment files with the actual fragments.
If the tolerance is specified it is interpreted in the chosen unit of length (units). The default tolerance is 0.001 Angstrom and the maximum is 0.1 a.u.

Input atomic coordinates that are slightly (within the tolerance) off from their correct positions are adjusted by the program.

Ghost Atoms & Non-standard Chemical Elements

The atom type names used under atoms (and in the create record) must begin with the standard chemical element symbol (H, He, Li...). The program uses this to deduce the nuclear charge and other elemental properties.

For the standard elements one can redefine the atomic mass (for instance to define a suitable isotope).

```
CREATE H {m=value} datafile
```

`value`

The atomic mass, which will then override the default value for the indicated chemical element.

A more extensive feature is available to define an artificial chemical element with user-specified properties. Such new elements are denoted *Alternative Elements*;and may for instance have a non-integer nuclear charge. The chemical symbol of an Alternative Elements is gh (for ghost) or J: either one is ok.

You can create j-type or gh-type basic atoms and use them subsequently as fragments in a molecule.

Starting from ADF2006.01 the BASIS key recognizes elements denoted with Gh.atom in the ATOMS key as being ghost atoms. If one does not specifically select a basis set for this ghost atom, the all electron basis set for the atom is selected in the creation of the ghost atom using the type of basis set chosen with the BASIS key. The atom name must begin with the standard one- or two-character symbol for the chemical element: Gh.H, Gh.He, Gh.Li, and so on. Optionally it may be appended by .text, where text is any string (not containing delimiters). Examples: Gh.H, Gh.Mn.3, Gh.Cu.dz-new.

Creation

The nuclear charge of an Alternative Element is not pre-defined, and *must* therefore be specified in the Create run. The atomic mass is optionally supplied.

```
CREATE J.NewElement q=Q {m=mass} datafile
```

`J.NewElement`

The atom type name, beginning with the *alternative* chemical element symbol J. It has an (optional) suffix of the form .text, completely similar to the construction of atom type names from standard chemical element

symbols.

`Q`

The nuclear charge. The q= option *must* be used for a J-element.
It must *not* be used for *standard* chemical elements.

`mass`

The atomic mass, in atomic mass units. If not supplied it will be set to the atomic mass of the standard chemical element with nuclear charge A, where A equals Q rounded to the nearest integer, but not smaller than 1 and not larger than 103.

`datafile`

The Create data file.

If you want to use the Alternative Element feature you'll have to construct your own Create data file, suited to the Alternative Element you have in mind. Appendix 1 describes the format of such a file.

Use as fragment

J-type basic atoms can be used like any other basic atoms to build up larger fragments and molecules. In fact, J (or Gh) can be considered just one more chemical symbol along with the 103 traditional ones. The element J has no pre-defined properties. Therefore you have to specify them where appropriate (c.f. the nuclear charge and atomic mass).

You may have different J-elements in a molecule, with different nuclear charges for instance. Yet, they must be denoted with the same chemical symbol J; the difference can only be made clear by the .text suffix in the atom type name.

It does no harm of course to make this suffix a concise but clear description of the main characteristics.

Basis Set Superposition Error (BSSE)

The Ghost Atom feature enables the calculation of Basis Set Superposition Errors (BSSE). The idea is as follows. In a normal calculation of the bonding energy of a molecule c, composed of fragments a and b, one compares the total energies of c vs. those of isolated a and isolated b added together. In adf this can be done in one stroke by running c from fragments a and b.

The BSSE is determined as the bonding energies of a pseudo-molecule d composed of (1) a plus a ghost b and (2) b plus a ghost a. The ghost atoms in the calculations are at their normal positions in the true molecule c, and they have their normal basis (and fit) functions. However, they do not have a nuclear charge and no electrons to contribute to the molecule. To set such a calculation up one needs first to make the appropriate ghost database files: for each involved atom, copy the database file that was used for its creation and modify it so as to remove the frozen core. Next, Create the ghosts with zero mass and zero nuclear charge. Apply these ghost fragments in the BSSE runs.

An example is worked out in the Examples document.

Hamiltonian

Spin-polarized start-up potential

The Coulomb and XC (exchange + correlation) potentials are computed from the fit approximation of the charge density (see Chapter 1.2).

The fit coefficients of this approximation for the first SCF cycle, needed to compute the first Fock matrix, are read from the fragment files: the start-up density is chosen as a sum-of-fragment-densities (fit approximations) and this combined density defines the initial potential.

In the SCF restart run the fit coefficients may be read from the attached TAPE21 file, see the key RESTART.

In some applications you may want to modify the initial fit coefficients (from the restart file or the fragment files). This is achieved with the key MODIFYSTARTPOTENTIAL. It allows you to scale them in some way so as to represent user-chosen amounts of spin-α and spin-β fit density on some or all of the fragments. This will adjust the spin-α and spin-β initial potentials.

This option applies only to *unrestricted* calculations of course. It may be used to help the program find a particular state. This might, for instance, be hard to find otherwise due to the a-b symmetry in the start-up situation. It may also be useful to speed up the SCF convergence in case you know what the final distribution of spin-α and spin-β density over the molecule will approximately be.

```
MODIFYSTARTPOTENTIAL {specification}
{ frag alfa // beta
 frag alfa // beta
 ...
end }
```

`ModifyStartPotential`

A *general* key: it has an argument *or* a data block.

`specification`

Must be *two numbers*, ASPIN and BSPIN, if provided at all. They specify the (relative) amounts of spin-α and spin-β fit density to define the spin-dependent potential at the first SCF cycle. The coefficients retrieved from the fragment files (or from the restart file in case of a SCF restart) are scaled accordingly. This will not affect the *total* amount of fit density: the absolute values of ASPIN and BSPIN play no role, only their ratio. In case of a restart run the restart file must have been generated in a *restricted* calculation, while the continuation run must be an *unrestricted* one.

If no argument is given a data block must be supplied with records frag alfa // beta.
This is very much similar to the main option with ASPIN and BSPIN: you specify ASPIN and BSPIN now separately for each fragment. This involves somewhat more input but increases the possibilities to tune the initial potential. Again this can be applied only in an unrestricted calculation. It cannot be used in a restart: the affected fit coefficients are those from the fragment files, while in an SCF restart run these are ignored and replaced by the coefficients on the TAPE21 restart file.
Each line specifies a frag with its corresponding ASPIN and BSPIN fit partitioning. If frag is the name of a fragment *type*, the specified ASPIN-BSPIN is applied to all individual fragments of that type. Alternatively an *individual* fragment can be specified, using the format fragtype/n, where *n* is an index between one and the total number of fragments of that type. In such a case the ASPIN-BSPIN data applies only to that particular fragment while different values may be supplied for the other fragments of the same type.
It is allowed to specify for certain fragment types individual fragments and for other fragment types only the type. Duplicate specifications are not allowed; an individual fragment must not be specified if its fragment type is also specified as a whole.
If the data block form is used, only the fit coefficients of the referenced fragments are affected. For the not-referenced fragments the fit densities are used as they are defined on the corresponding fragment files.

The SCF convergence of a spin-unrestricted calculation usually improves when you start with potentials that correspond to the correct ratio of spin-α and spin-β electrons. By default ASPIN=BSPIN=0.5, as implied by

the spin-restricted start density of the fragments or restricted molecule.

The total amount of fit density used on the first iteration is defined by the sum-of-fragment densities (or the density on the restart file). This may be different from the total nr. of electrons in the actual calculation. On the second SCF cycle the fit density will internally be normalized so as to represent the correct number of electrons.

The block-form of the key makes the start up of broken symmetry calculations easy. For example one may want to start a calculation in broken symmetry with spin-α density on one fragment and spin-β density on another, e.g. in a spin-unrestricted calculation of H2 at large separation. It is particularly useful for larger systems, e.g. for magnetic coupling between spin-polarized magnetic centers, as in Fe-S complexes [111]: start with oppositely polarized Fe centers, but with, for instance, the remaining bridge and terminal ligands unpolarized. See also the N2+ sample run in the examples.

Unrestricted fragments

The fragments from which the molecule is built must be spin-restricted, that is: the fragment files must be result files of spin-restricted calculations. For purposes of analysis, however, it may be desirable in some applications to build your molecule from *un*restricted fragments. This can be simulated as follows.

You tell adf that you want to *treat* the fragments as if they were unrestricted; this causes the program to duplicate the one-electron orbitals of the fragment: one set for spin-α and one set for spin-β. You can then specify occupation numbers for these spin-unrestricted fragments, and occupy spin-α orbitals differently from spin-β orbitals.

Of course, the unrestricted fragments that you use in this way, are not self-consistent: different numbers of spin-α and spin-β electrons usually result in different spatial orbitals and different energy eigenvalues for spin-α and spin-β when you go to self-consistency, while here you have spatially identical fragment orbitals. Nevertheless it is often a fair approximation which gives you a considerable extension of analysis possibilities.

```
FRAGOCCUPATIONS
 fragtype
   irrep spin-a // spin-b
   irrep spin-a // spin-b
   ...
 subend
 fragtype
   irrep spin-a // spin-b
   ...
 subend
 ...
end
```

`fragtype`

One of the fragment types and functions as a (block type) subkey. The data block for the subkey ends with the standard end code for block type subkeys (subend).

`irrep`

One of the irreducible representations (irreps) for the point group symmetry that was used in the computation of that fragment.

`spin-a // spin-b`

Two sequences of occupation numbers, which will be applied to the spin-α and spin-β versions of the Fragment Orbitals. The sequences must be separated by a double slash (//). See for comparison the specification of occupation numbers for the overall system (key CHARGE).

The sum of spin-α and spin-β occupations must, for each fragment orbital in each irrep separately, be equal to the total (restricted) occupation of that orbital as it is stored on the fragment file. In other words: you can only change the distribution over spin-α and spin-β electrons within one orbital.

(Without this restriction the spatial distribution of the total (sum over spins) fragment charge density would be changed, leading to an incorrect bonding energy analysis after the calculation).

The data block of fragoccupations is not parsed for expressions and constants or functions defined under define. Any such items will not be recognized and not be replaced by their values.

Be aware that in more-dimensional irreps (e, t) the number of electrons in a fully occupied orbital is input as the dimension of the irrep times the one-electron orbital occupation. Compare the key OCCUPATIONS.

For irreps that are not mentioned in this input block, and hence for all irreps of fragment(type)s that are not mentioned at all, the spin-α and spin-β occupations will be set equal, which is of course what they in fact are on the (restricted) fragment file.

For an example of applying this option see [112].

Remove Fragment Orbitals

By default all fragment orbitals (the MOs of the fragment computation), which are stored on the fragment file, are used as basis functions for the overall molecule, see Chapter 1.2. You can remove one or more of these fragment orbitals from the basis set of the molecule. This may be useful for special analyses, for instance to study the effect of deleting all virtual MOs of a particular fragment (CSOV analysis, Constrained Space Orbital Variation). It may also enhance the efficiency since you effectively reduce the size of the basis set, but you should be aware of the potential effects on the results.

```
REMOVEFRAGORBITALS
 fragtype
   subspecies nremove
   subspecies nremove
   ...
 subend
 fragtype
   subspecies nremove
   ...
 subend
 ....
 (etc.)
 ....
end
```

`fragtype`

One of the fragment types in the system. Any subset of the available fragment types can be used here as subkey. The subkeys are block type keys; their data blocks end subend.

`subspecies`

One of the subspecies of the irreducible representations of the point group symmetry that was used in the calculation of the fragment itself. This requires of course that one knows the symmetry that has been used

for the fragment calculation.

`nremove`

The number of fragment orbitals of the pertaining representation that will not be used as basis functions for the overall system. The *highest* (in energy eigenvalue) nremove orbitals are discarded. You must not remove *occupied* fragment orbitals.

By default (omission of the key) all fragment orbitals are used in the basis set for the system.

Important Note

It is imperative that any removal of fragment orbitals will not break the symmetry of the molecule. This consideration is relevant when for instance two different subspecies of a fragment irrep contribute to different partner subspecies in one of the irreps of the molecule. In such a case, when one removes an orbital in such a fragment subspecies, its partner orbital should also be removed. If this is violated an error may occur or the results will simply be wrong. Quite likely, the program will detect the error, but this may occur only in the final (analysis) stage of the calculation so that a lot of CPU time may have been wasted.

Example: consider a single-atom fragment, computed in atom symmetry, used as fragment in a c(lin) molecule and assume that the p:x and p:y fragment orbitals contribute to respectively the pi:x and pi:y subspecies of the molecule. Then, when you remove one or more p:x fragment orbitals, you should also remove the same number of p:y fragment orbitals. Practical cases may be more complicated and whenever you use this key, make sure that you've fully analyzed and understood how the fragment irreps combine into the molecular symmetry representations. Hint: run the molecule, without removing any fragment orbitals, and stop at an early stage after the program has computed and printed the build-up of the molecular SFOs from the fragment orbitals. To control early aborts via input, use the key STOPAFTER.

Core Potentials

In the standard approach the Coulomb potential and the charge density due to the atomic frozen core are computed from the frozen one-electron orbitals. adf stores the computed core density and core potential for each atom type in the molecule on a file TAPE12. Alternatively, you may attach a file with (core) potentials and densities. The file must have the same structure as the standard TAPE12. It should contain one or more sections, each with the core information for one type of atom. With the key COREPOTENTIALS you specify the core file and (optionally) which sections pertain to the distinct atom types in the molecule. It is a general key that can be used as a simple key or as a block key.

```
COREPOTENTIALS corefile { &
  atomtype index
  atomtype index
  ...
end}
```

`corefile`

The file with core potentials and charge densities. The name may contain a path.

`atomtype`

One of the atom type names as defined by atoms.

`index`

Points to the core section on the attached file that applies to the atom type. Different atom types may use the same section. A non-positive index tells the program that the atoms of that type don't have a frozen core. If the information on the corresponding fragment file (or data file in Create mode) indicates the contrary

the program will abort with an error message.

If the key is used as a simple key (specifying only the core file) the sections on the file are associated with the atom types in order: the first section is used for the first atom type, et cetera. This is overruled by applying the block form. However, since the key *must* have the core file as argument, the block form requires that you apply the continuation symbol: an ampersand (&), separated from the core file name by at least one blank.

If you omit an atom type from the data block it gets a zero index (no core).

The attached file may contain more sections than used in the calculation, and the indices specified in the data block don't have to be in ascending order, consecutive, or cover a specific interval.

When a file with non-standard (e.g. relativistic) cores is attached and used in the calculation of an atom or molecule, and the result is used as fragment in a subsequent calculation, you should attach and use the same core potentials again. Otherwise, the program will internally compute the standard core potentials and hence implicitly employ another fragment than you may think, i.e. a fragment with other properties. adf will not check anything in this respect and corepotentials should therefore be handled with great care.

The primary application of the corepotentials option is to include (scalar) relativistic corrections in the (frozen core part of the) Fock operator. The relativistic core potentials can be computed with the auxiliary program dirac (see the utilities document)

Properties and Analysis

NMR Chemical Shifts

NMR Chemical shifts have been implemented [113-117] in a separate property program NMR. It requires the TAPE21 result file from an ADF calculation. See the 'ADF Property Programs' document for the input description of the NMR module. The NMR module can be combined with the ZORA treatment for relativistic effects and with Spin-Orbit effects, making it suitable for treatment of heavy elements.

Note: NMR calculations on systems computed with Spin-Orbit relativistic effects can only be performed by the NMR module if the ADF calculation has suppressed usage of symmetry, i.e. when the symmetry used in the ADF calculation has been NOSYM.

The program CLGEPR (or simply 'EPR'), described separately, also contains NMR functionality. This NMR implementation provides a detailed breakdown of the orbital contributions to the calculated quantities. This can be of use for analysis purposes. Again, please check the separate 'ADF Property Programs' documentation for details.

Warning: the NMR and EPR property program will not always give the correct result for every SCF potential in the ADF calculation, like for example the SAOP potential, or if one uses COSMO in the ADF calculation. This is due to the GIAO method used in these property programs, which requires the calculation of the SCF potential, which is not done correctly for potentials, other than the standard LDA and GGA potentials. To obtain correct results one should, in addition to the use of TAPE21, also use TAPE10 that ADF generates, using the keywords SAVE TAPE10, and use it as input for the NMR or EPR property program. On this TAPE10 the SCF potential is written. See also the separate 'ADF Property Programs' documentation.

NMR spin-spin coupling constants

The NMR spin-spin coupling constants [118, 119] have been implemented in a separate program CPL . It can be combined with ZORA and Spin-Orbit treatment of relativistic effects to study heavy elements. See the

separate 'ADF Property Programs' documentation for this property.

EPR parameters

A separate program CLGEPR (or simply EPR) supports calculations of Electron Paramagnetic Resonance (EPR) g-tensors of closed-shell and open-shell molecules[120, 121]. Low-spin and high-spin EPR g-tensors can both be calculated. Both non-relativistic and scalar Pauli Hamiltonians are supported. Spin-orbit and ZORA are not supported however. A detailed breakdown of the orbital contributions can be provided on output. Please check the separate 'ADF Property Programs' documentation for details and also compare to the functionality of the ESR calculation described in this manual (ESR and QTENS keywords) to find the most suitable approach for your problem.

Warning: the NMR and EPR property program will not always give the correct result for every SCF potential in the ADF calculation, like for example the SAOP potential, or if one uses COSMO in the ADF calculation. This is due to the GIAO method used in these property programs, which requires the calculation of the SCF potential, which is not done correctly for potentials, other than the standard LDA and GGA potentials. To obtain correct results one should, in addition to the use of TAPE21, also use TAPE10 that ADF generates, using the keywords SAVE TAPE10, and use it as input for the NMR or EPR property program. On this TAPE10 the SCF potential is written. See also the separate 'ADF Property Programs' documentation.

Localized Molecular Orbitals

ADF provides the Boys-Foster method for localization of Molecular Orbitals [122-124]. This implies a unitary transformation of the occupied molecular orbitals as computed in the SCF procedure, with the objective to obtain a (transformed) set of orbitals that represent exactly the same charge density but with molecular orbitals that are more localized in space than the original MOs.

The goal of orbital-localization lies in analysis: the localized orbitals provide an easier-to-interpret picture.

Orbital localization procedures require a measure of the localization of the orbitals which can then be optimized in the space of the allowed unitary transformations. Methods advocated in the literature differ in the definition of this measure. The Boys-Foster method minimizes the mean extension of the occupied orbitals around their center of gravity; see the literature for details.

Occasionally it is useful to apply the localization only to a subset of the MOs, with the objective to expose certain features better. This is accomplished by performing the localization in a number of distinct steps, where at each step the localization is restricted by keeping a subset of the MOs frozen. A case is worked out in the Examples document.

The computation of localized orbitals is controlled with the block-type key . By default (if the key is not supplied in input) no orbital localization is carried out.

```
LOCORB {nopop store}
  Spintype FrozenMOs
  Spintype FrozenMOs
  ...
end
```

nopop

Specifies that no SFO population analysis is to be carried out on the localized MOs. By default this population analysis will be printed in the output file.

store

Specifies that the transformation from MOs to localized MOs is stored on TAPE21.

`Spintype`

Must be either alfa or beta (not case sensitive) and refers to spin-A and spin-B orbitals respectively. In a spin-restricted run beta records are meaningless and must not be used.

`FrozenMOs`

A list (possibly empty) of integers, referring to a list of MOs from the SCF, and/or labels of irreducible representations. The integers and/or labels may be given in any order.

Each record Spintype FrozenMOs in the data block defines a localization *cycle* in which the localization procedure is carried out on all orbitals (of the indicated spin), except those indicated by the FrozenMOs.

For either spin at least one localization cycle is carried out. If no data record for that spin is found in the data block, a full localization is performed, without any MOs excluded.

The data block may be completely empty (but the record end must be supplied since the key is block-type) and would be equivalent with specifying two records, one for either spin, without any FrozenMOs:

```
LOCORB {nopop}
end
```

is equivalent with

```
LOCORB {nopop}
  alfa
  beta
end
```

The integers in FrozenMOs refer to an overall list of SCF MOs consisting of all valence MOs in each symmetry representation up to and including the highest non-empty one. So, when for instance in the first irrep MO #4 is the highest non-empty one and in the second irrep mo #2 is the highest non-empty one, then in the overall list the first 4 are the orbitals of the first irrep, the no.s 5 and 6 are from the second irrep, etcetera.

Each symmetry label in FrozenMOs collectively denotes in one stroke all molecular orbitals of that representation up to and including the highest occupied one (in that symmetry). The label may be the name of an irreducible representation or of a subspecies. In the former case all partner representations are denoted collectively. In an atom symmetry for instance, specifying P would be equivalent to P:x P:y P:z.

Note that if the final SCF has in any symmetry representation empty orbitals *below* the highest non-empty orbital in that symmetry - violating the Aufbau principle - then these empty orbitals are included in the above-defined overall list and hence a FrozenMOs specification is necessary, namely to avoid mixing MOs with different occupation numbers in the localization.

Note:

It is imperative that in a particular localization cycle only MOs from the SCF are combined that have identical occupation numbers. If this is violated the program will carry out the localization without error message, but the results are incorrect in the sense that the density defined by the localized orbitals is *not* the same anymore as the SCF density.

So, if any of the MOs in the overall list defined above is not *fully* occupied (open shell, excited state, ...) you need to define precisely the localization cycles - localizing in each cycle only MOs with identical occupations and freezing all others - in order to obtain sensible results.

In the output file the localized MOs are printed as expansions in SFOs and (optionally) a population analysis

is given, again in terms of the SFOs. Furthermore, each localized MO has associated with it an energy value and an occupation number. The energy is the expectation value of the Fock operator for the orbital. The occupation number is obtained as a weighted sum from the SCF MOs that were combined into the localized orbital. As mentioned before one should combine only SCF MOs with identical occupations into a localized orbital, in which case its occupation number will be the same. The printout of the occupation number of the localized orbital allows therefore a verification that a correct localization procedure has been carried out.

Bond order analysis

Bond order analysis in ADF is activated by keyword

```
BONDORDER {tol=xxx} {printall}
```

By default bond order indices calculated by the Nalewajski-Mrozek [148-152] method are calculated. There exist three alternative definitions of the valence and bond order indices within the Nalewajski-Mrozek approach. By default the values obtained from partitioning of Tr($P\Delta P$) are calculated and printed in the output. For more information on alternative Nalewajski-Mrozek bond order indices see Results/Properties section (5.1).

`tol=xxx`

The tol=xxx option specifies the threshold value for bond orders to be printed in the output (default=0.2).

`printall`

The values calculated from all three versions of the Nalewajski-Mrozek approach are printed when the option printall is present; in addition the Gopinathan-Jug [153] and Mayer [140] bond order indices are calculated for comparison.

Present bond order analysis is based on SFOs. Symmetry used in the calculation should be NOSYM. For this reason the analysis may be used only if the symmetry in the calculation is NOSYM. The analysis may be used also for multi-atomic fragments, the fragment-fragment bond orders are printed in such a case. Note that in the present implementation all fragment types should be different.

NBO analysis

The adf release contains a simple input file generator, called adfnbo, for the GENNBO program of Prof. Weinholds Natural Bond Orbital (NBO) 5.0 package:

http://www.chem.wisc.edu/~nbo5

The GENNBO executable is included in the ADF distribution and can be enabled via the license file for all those who buy an NBO manual from SCM. An extensive documentation of GENNBO is part of the NBO manual. Usage can be found in the Analysis and Examples Document.

Bader's analysis

ADF utility adf2aim (original name rdt21) developed by Xavi López, Engelber Sans and Carles Bo (see http://www.quimica.urv.es/ADF_UTIL): convert an ADF TAPE21 to WFN format (for Bader analysis)

The program rdt21 is now called adf2aim and is part of the ADF package, starting from ADF2004.01.

The WFN file is an input file for the third party program Xaim (see http://www.quimica.urv.es/XAIM for

details), which is a graphical user interface to programs that can perform the Bader analysis. Usage of adf2aim can be found in the Analysis and Examples Document.

Precision

Numerical integration

The key INTEGRATION has been introduced in its simple form in Chapter 2.2.

```
INTEGRATION accint
```

accint is a real number. The key is used as a simple key here.

Alternatively you can use it as a block key. This is activated if you give no argument. In the data block you specify which of several integration methods you want to use, and you give values for the involved parameters. Consult the literature for detailed information about the various schemes.

```
INTEGRATION
  data
  data
  ...
end
```

The block form is used to override default relations between various parameters that are applied in the generation of the integration grid in the polyhedron method [105]. All these parameters are accessible with subkeys in the data block of Integration. Most of the subkeys are simple keys with one single value as argument; a few subkeys are block-type (sub) keys themselves and hence require the usual format of a data block closed by subend.

`accint`

The main precision parameter
Its value defines the number of significant digits by which an internal set of standard integrals must be evaluated. The number and distribution of integration points is tuned accordingly. For normal applications this should yield a nearly optimal (given the underlying method) generation of points and weights. The default depends on the run type.

`accsph`

The polyhedron method of generating integration points partitions space in atomic polyhedrons, partitioned in pyramids with their tops at the atom in the center of the polyhedron. A core like atomic sphere is constructed around the atom; this truncates the tops of the pyramids. accsph specifies the test precision for the generation of points within the spheres. By default accsph=accint.

`accpyr`

Similarly this subkey sets the test level for the parts of the pyramids outside the atomic sphere. Default: accpyr=accint.

`accpyu, accpyv, accpyw`

The truncated pyramids are mathematically transformed into unit cubes. A product Gauss integration formula is applied to the cubes, with three (test precision) parameters for the three dimensions. Accpyw controls the direction that is essentially the radial integration from the surface of the atomic sphere to the base of the pyramid. The other two control the orthogonal directions (angular). By default all three equal accpyr.

`accout`

The region of space further away from the atoms, outside the polyhedrons, has its own precision parameter. By default accout=accint.

`nouter`

This outer region is treated by a product formula: outwards times parallel. The latter involves two dimensions: the surface of the molecule say. The outward integration is performed with Gauss-Legendre quadrature, in a few separate steps. The lengths of the steps are not equal, they increase by constant factors. The total length is fixed. The number of steps is controlled with this subkey; default: 2.

`outrad`

The parameter that defines the number of Gauss-Legendre integration points for each outward step. The precise relation between the actual number of points and this subkey, and the default relation between outrad and accout can be found in the implementation.

`outpar`

Similarly the integration in the directions parallel to the surface of the atomic system is controlled by a parameter. See the implementation for details.

`dishul`

Sets the distance between the outermost nuclei of the molecule and the boundary planes that define the boundary between the polyhedrons and the outer region. By default dishul=2.3*R, where Ris the radius of the largest atomic sphere in the molecule.

`frange`

The outward range of the *outer region*: integration is not performed to infinity but to a distance frange from the outermost atoms, where all functions can be assumed to be essentially zero. By default frange is derived both from accint, the general precision parameter, and from the present chemical elements: heavier atoms have longer-range functions than hydrogen say. The precise relations can be found in the implementation.

`linrot`

This parameter is significant only for symmetries with an axis of infinite rotational symmetry: Cand D
It is the highest rotational quantum number around this axis that occurs among the integrands. This depends on the employed basis functions and fit functions. By default the program finds this out for itself.

`qpnear`

If you specify point charges in the input file, there are two considerations implied for the numerical integration grid.
First, since the point charges create a Coulomb singularity. The integrands (of for instance the basis function products against the Coulomb potential) can only be evaluated with high precision if the grid around the point charges has spherical symmetry and uses local spherical coordinates, exactly as is done for the atomic nuclei. Second, the point charges do not carry fit or basis functions, hence they play only a role in the more diffuse tails of the actual functions involved in integrals. Therefore, a relative low precision of the integral part close to the point charge may have little effect on the total integration accuracy.
Since additional 'spherical centers' with their own surrounding grids increase the total number of points significantly, typically a few thousands *per Coulomb center*, this may result in high computational effort. Therefore, the program generates spherical grids only about those point charges that are close to the other atoms. The criterion, input with the qpnear subkey, is the closest distance between the point charge at hand and any real atom. Default 4.0 Angstrom. Any input value is interpreted in the unit-of-length specified with

the Units key.

Next come the subkeys that require a list of data. The subkey must be placed on one line, the data on the next. This somewhat peculiar structure suggests that the subkeys are block keys; however their data blocks have no end code (subend) as for normal block type subkeys.

The list of data for such a subkey contains one value for each atom type. The data must be in the order in which the atom types were defined under atoms, implicitly or explicitly: remember that atoms belonging to different fragment types automatically have different atom types, even if their atom type *names* have been specified as identical under atoms.

`rspher`

gives the radii of the atomic spheres, one value for each atom type. By default, the radii are derived from the chemical element (heavier atoms get larger spheres) and from the environment: the sphere must not be too large for the atomic cell (polyhedron).

`linteg`

The maximum angular momentum quantum number of integrands centered on an atom of that type (one value for each atom type). This depends on the basis functions and on the fit functions. By default the program checks the function sets and sets the linteg values accordingly. This subkey is applied for the generation of grid points in the atomic spheres.

Items that relate to geometric lengths (dishul, frange, rspher) must be given in bohr (=atomic units), irrespective of the unit of length defined with units.

Symmetric density fit

The density fitting procedure in adf is carried out separately for each pair of atoms. The implemented approach has several advantages in efficiency but it has a drawback in that it necessitates the use of all available fit functions rather than only the symmetric combinations although the final result of course needs only a symmetric fit because the total density is a symmetric (A1) function. For atoms far apart the density fitting is performed with only symmetric functions. Given the implemented algorithm this entails an approximation which can be tuned:

```
A1FIT atomicseparation
```

`atomicseparation`

is the threshold distance between atoms, *in Angstrom*. The symmetric fit approximation is applied only for atoms farther apart. Default is 10.0 Angstrom

Fit integrals

For the computation of the Coulomb potential the program uses a large number of so-called *fit integrals*: the overlap integrals of a fit function with a *product* of two basis functions, where at least two of the involved three functions are centered on the same atom. In fact these are ordinary overlap integrals of STOs because the fit and basis functions are all STOs and a product of STOs on a center is itself also an STO.

Obviously, when the two involved atoms are far enough apart, such overlap integrals become negligibly small. All fit integrals are ignored (and not computed) that are smaller - according to a rough but reasonable estimate - than a preset threshold.

The value of this treshold can be set via input:

```
EPSFIT accfitint
```

The threshold for ignoring fit integrals is 10-accfitint. The default for accfitint is 4.0.

True density in XC potential

For the computation of the exchange-correlation potential (XC-potential) the program uses as default the fitted density. This is an approximation. For the XC potential the true density can be used if one includes the keyword EXACTDENSITY:

```
EXACTDENSITY
```

Atomic radial grid

For each atom the charge densities and the coulomb potentials of frozen core and valence electrons are computed in a radial grid and stored on TAPE21. The values in the points of the molecular numerical integration grid are then evaluated by interpolation from the table of radial values.

The radial grid consists of a sequence of r-values, defined by a smallest value, a constant multiplication factor to obtain each successive r-value, and the total number of points. Equivalently it can be characterized by the smallest r-value, the largest r-value, and the number of points; from these data the program computes then the constant multiplication factor. The characteristics are set with

```
RADIALCOREGRID {nrad=points} {rmin=rmin} {rmax=rmax}
```

points

The number of radial grid points; default: 5000.

rmin

The shortest distance used in the radial grid; default 1e-6 Angstrom

rmax

The largest distance in the radial grid; default: 100 Angstrom.

rmin and rmax, when specified, are interpreted as specified in units of length defined by units.

The keyword name radialcoregrid has historical reasons: in earlier releases the radial grid was used only for the frozen core density and potential.

Dependency (basis set, fit set)

Conceivably the sizes of basis and/or fit sets may be so large that the function sets become almost linearly dependent. Numerical problems arise when this happens and results get seriously affected (a strong indication that something is wrong is if the core orbital energies are shifted significantly from their values in normal basis sets). Although for the fit set a few (incomplete) tests are carried out, the program will generally not check such aspects and carry on without noticing that results may be unreliable.

A new feature has been implemented to take care of this. For reasons of compatibility with previous versions and also because our experience with it is limited so far, we have chosen to make application of it not the default.

You have to activate it explicitly. Our experience so far suggests that real problems only arise in case of large basis sets with very diffuse functions (i.e.: not with the normal basis sets provided in the standard package).

Use of the key DEPENDENCY turns internal checks on and invokes countermeasures by the program when the situation is suspect. A few technical (threshold-type) parameters can be set as well, but this is not necessary, assuming that the defaults are adequate.

```
DEPENDENCY {bas=tolbas} {eig=BigEig} {fit=tolfit}
```

`tolbas`

A criterion applied to the overlap matrix of unoccupied normalized SFOs. Eigenvectors corresponding to smaller eigenvalues are eliminated from the valence space. Default value: 1e-4. Note: if you choose a very coarse value, you'll remove too many degrees of freedom in the basis set, while if you choose it too strict, the numerical problems may not be countered adequately.

`BigEig`

Merely a technical parameter. When the DEPENDENCY key is activated, any rejected basis functions (i.e.: linear combinations that correspond with small eigenvalues in the virtual SFOs overlap matrix) are normally processed until diagonalization of the Fock matrix takes place. At that point, all matrix elements corresponding to rejected functions are set to zero (off-diagonal) and BigEig (diagonal). Default: 1e8.

`tolfit`

Similar to tolbas. The criterion is now applied to the overlap matrix of fit functions. The fit *coefficients*, which give the approximate expansion of the charge density in terms of the fit functions (for the evaluation of the coulomb potential) are set to zero for fit functions (i.e.: combinations of) corresponding to small-eigenvalue eigenvectors of the fit overlap matrix. Default 1e-10.

Notes:

- Application / adjustment of tolfit is not recommended: it will seriously increase the cpu usage while the dependency problems with the fit set are usually not so serious anyway.
- Application of the dependency/tolbas feature should not be done in an automatic way: one should test and compare results obtained with different values: some systems look much more sensitive than others.
  We have, so far, not been able to understand an unambiguous pattern in these experiences. Of course, when things become clearer in this respect, we will implement the corresponding intelligence into the program.
- When the dependency key is used, the numbers of functions that are effectively deleted is printed in the output file, in the SCF part (cycle 1) of the computation section.
- The TAPE21 result file of a calculation that used the DEPENDENCY key contains information about the omitted
  functions and these will also be omitted from the fragment basis when the TAPE21 is used as a fragment file.

Control of Program Flow

Limited execution

```
STOPAFTER programpart
```

`programpart`

Must be a predefined name associated with a (major) part of the program With this key you tell adf to terminate the job after the named program part has been executed.

A survey of the recognized names with a brief explanation follows below. The program parts are listed in order of execution: by taking a name further down the list you execute a larger part of the program.

`init`

initialization procedure, input reading and printing of the output header with the job identification.

`input`

input-reading module.

`geomet`

geometry section: organization of atoms in types of atoms and fragments, checks of the actual fragments against information on the attached fragment files.

`config`

electronic configuration (if not determined only by the SCF procedure), printout of symmetry subspecies.

`mainsy`

generation of symmetry information, representation matrices, etc.

`symfit`

construction of symmetry adapted fit functions.

`cblock`

generation of integration points and the distribution of them in the blocks that control the internally used segmented vectorization loops.

`engrad`

Relevant only in an optimization calculation. Engrad calculates energy gradients. The geometry is not yet updated and no printing of convergence tests and new coordinates is carried out.

`geopt`

This routine evaluates energy gradients and updates the geometry accordingly; it also prints the convergence tests and the computed new coordinates. Compare 'stopafter engrad'.

`forcematrix`

in a Frequencies run, terminate the calculation when all displacements have been done and before any further processing of the computed hessian, such as the determination of normal modes, takes place.

Direct SCF: I/O vs. recalculation of data

The program's performance can be defined in terms of the amounts of time (cpu and i/o seconds) and disk space used in a calculation. Also important for the human user is the turn-around time. On multi-user machines cpu-cheap jobs may take a lot of real time to execute due to i/o scheduling.

Therefore it can be a good idea to recompute some items rather than store them on disk. This will increase the amount of cpu time but reduce disk access and it may also improve the turn-around. Another

consideration is of course that storage of data on disk may exhaust the available disk space in case of big calculations so that recalculation rather than storage is unavoidable.

```
DISK {{no}fit} {{no}basis}
```

instructs adf how to handle the values of the fit functions and basis functions in all integration points: calculate once and store on disk or recompute whenever needed. The (optional) arguments are fit or nofit, and basis or nobasis.

fit and basis tell adf to store the corresponding data on file; the prefix no induces recalculation whenever the data is needed.

Defaults are nofit and nobasis: direct-SCF mode for both features (this can be modified at the installation of adf, see the Installation Manual).

The key DISK has replaced in adf 2.0 the key directSCF in adf 1.x, and extended the applicability of the I/O versus recalculation choice from fit functions-only to basis functions as well.

Skipping

With the following key you can restrict which parts of the program are actually executed:

```
SKIP argumentlist
```

`argumentlist`

A sequence of names, separated by blanks or commas. skip may occur any number of times in input. The names in the argument list refer to various items that are associated with parts of the program. With this key you tell adf to skip the named program part(s) and to continue execution thereafter. The program does not check any consequences and may even crash when variables have not been initialized or have attained incorrect values due to the skipping.

Use of this key should be contemplated only in debugging and testing sessions, in which you may skip the computation of certain data when before that data will be needed you'll halt the program to inspect something.

Recognized and operational arguments are for instance (possibly not complete due to frequent extensions in this respect): atpair, ets, fitint, orthon, qmpot

Ignore checks

adf performs several checks during a calculation, and stops with an error message when intermediate results are suspicious, when input-specified instructions are incompatible, etc. These controlled aborts can in some cases be overruled. Of course, the checks have been inserted for good reasons and one should realize that ignoring them probably produces incorrect results and/or may lead to a program-crash.

```
ALLOW argumentlist
```

`argumentlist`

A sequence of names, separated by blanks or commas. allow may occur any number of times in input, see the list below for the names that can be used.

`BadCoreInt`

Numerical integration of the frozen core density should closely approximate the analytical value. If the

deviation is large compared to the user-specified numerical integration precision the program aborts with an error message like 'BAD CORE INTEGRAL'. This control is overruled by using this ALLOW option.

`BadIntegrals`

Only applicable when the direct-SCF option is turned off for the basis functions. (This happens automatically for ZORA full-potential calculations). In that case, a sequence of elementary overlap integrals are evaluated with the numerical integration grid and the outcomes tested against the analytical value. If the deviation is too large a warning is issued. Above a certain threshold the program will abort, unless you override the exit with this Allow option.

`BadSCF`

If the SCF procedure hasn't converged, any geometry manipulations (optimization, linear transit ...) will be aborted because the energy gradients are not reliably computed in a non-self-consistent field.

`CloseAtoms`

Atom-atom distances should not be less than 0.2 Bohr. This is checked in the program section where the numerical integration grid is generated.

`RelGeo`

Geometry manipulation (optimization, linear transit...) is not supported for all of the relativistic options. See Relativistic

`SmallBlocks`

The list of numerical integration points is partitioned in blocks, so as to fit data arrays (for instance values of all basis functions in the points of a block) in available memory. The program computes the maximum block length from available memory and size parameters such as numbers of basis functions. A small block size implies a severe reduction in CPU efficiency. Therefore, the program aborts (by default, to override by this ALLOW option) if the block length turns out to be very small (less than 10).

`XC`

Certain combinations of the Density Functional options or application of them with some other features are not allowed. See XC.

Parallel Communication Timings

With the key

```
COMMTIMING
```

in the input you instruct ADF to skip normal execution and perform only a test on the gather, broadcast and combine routines, used in a PVM version of ADF. Obviously, this is only meaningful if such an ADF version has been installed.

Technical Settings

Memory usage

The amount of memory used by the program during a calculation is determined by three quantities:

- The size of the program itself (executable statements, fixed arrays in subroutines...).
  This quantity depends on the program release number and is currently somewhere between 10 and 16 Mb.
- Buffer space used by adf for more efficient I/O handling. This quantity is set at installation.
  See the Installation Manual.
- Dynamically allocated arrays. The program allocates memory dynamically during the run conform the requirements of the actual calculation.

Vector length

Numerical integration is applied in ADF to evaluate Fock matrix elements and many other quantities that are defined as integrals over basis functions, the charge density, the potential, etc. As a consequence a large part of the CPU time is spent in simple do-loops over the integration points. The total number of points depends on the required precision and on the number of atoms, the geometry and symmetry. All such numerical integration loops are segmented into loops over *blocks* of points, each block consisting of a certain number of points. This latter defines the most inner do-loop and hence determines vectorization aspects.

Depending on the computer, c.f. the compiler, vector operations may be executed more efficiently using longer vectors. Long vectors increase the demand on Central Memory however because the program may sometimes have to access large numbers of such vectors in combination (for instance all basis functions) so that they must be available in memory simultaneously. The optimum vector length depends therefore on the balance between vectorization efficiency and memory usage. The maximum vector length that you allow the program to use can be set via input.

```
VECTORLENGTH vectorlength
```

The default is set at the installation of adf on your platform, see the Installation manual. For organizational reasons the true vector length actually used in the computation may be smaller than the value defined with this key, but will not exceed it (except in a Create run, but in that case performance and memory usage are no hot topics).

Tails and old gradients

The key TAILS is currently obsolescent because of the introduction of the LINEARSCALING keyword and may be removed in future versions. The key TAILS was used in older versions, ADF2004.01 and before, in the calculation of the gradients.

Each block of points (see above) covers (more or less) a certain region in space and can hence be assigned a distance value with respect to a particular atom. These distances are used to control whether or not to evaluate functions centered on that atom in that particular block of points.

```
TAILS {bas=tailbas} {fit=tailfit}
```

`tailbas, tailfit`

Accuracy levels, similar to the integration parameter: a higher value implies higher precision: in this case, basisfunctions and fitfunctions respectively are assumed zero in blocks of points that are at a sufficiently large distance from the atom at which the function is centered. Sufficiently large is defined by comparing the integral of the (radial part of the) function beyond that distance with the total integral. By default tailbas and tailfit both depend on the numerical integration parameter

Note: in contrast with some of the older versions, supplying only the keyword without parameters does not switch off the use of function cutoffs. To effectively switch off the distance effects in gradients evaluation

one should specify large values for the BAS and FIT parameters. The value of 100 should be more than enough, thus, for example:

```
TAILS bas=100 fit=100
```

Improved performance in geometry optimizations and frequency runs is achieved by a new implementation of the calculation of the gradients that now uses linear scaling techniques. This is now the default. One can still use the old implementation if one includes in the input:

```
OLDGRADIENTS
```

The key TAILS is not used in geometry optimizations anymore. For controlling the use of distance effects in normal SCF calculations, and for calculations with the RESPONSE or EXCITATIONS keywords, please check the LINEARSCALING keyword.

Linearscaling

The LINEARSCALING keyword has a very similar function to the TAILS keyword described above. In addition to defining the precision of operations related to operations in the numerical integration grid, it also defines the precision for the calculation of the overlap matrix, the fit integrals, and the density fit procedure. Default values have been chosen which result in negligible differences in the results for our test calculations, so that these defaults can be considered safe. They have been chosen similarly to the defaults for the TAILS keyword.

However, it may be advisable to modify the settings for the linear scaling parameters in two cases. First, if a very accurate result is needed, and numerical noise is to be completely eliminated, strict values can be specified. Especially for small molecules, where timings are not so large anyway, this may be of interest. Second, for large molecules, in which the calculations are very time-consuming, one can experiment with less strict values for the LINEARSCALING block keyword. In such a case one should be aware of the reduced accuracy and preferably test the influence of the changes on the results.

In the simplest application of the LINEARSCALING keyword, only one parameter is provided. All the subkeys described below will then be given this value. A very large value implies a calculation where no distance cut-offs are used. A normal value (almost default situation) would be 8 for linscal, 6 gives a faster but somewhat sloppier result. Whether this is acceptable is strongly case-dependent. A value of 10 or 12 is already quite strict and, unless there are some sort of numerical problems, there should not be much influence on the results by choosing a stricter value than that. A value of 99 for linscal virtually excludes the possibility that something will be neglected.

```
LINEARSCALING linscal
```

More refined control is possible by using the full block key

```
LINEARSCALING
  CUTOFF_FIT epsfit
  OVERLAP_INT ovint
  PROGCONV progconv
  CUTOFF_COULOMB epsvc
  CUTOFF_MULTIPOLES epsmp
END
```

CUTOFF_FIT

determines how many atom pairs are taken into account in the calculation of the fit integrals and the density fit procedure. If the value is too low, charge will not be conserved and the density fitting procedure will become unreliable. This parameter is relevant for the timings of the FITINT and RHOFIH routines of ADF.

The default value for epsfit is accint + 4 (typically 8) and, where accint is the value specified for numerical integration accuracy (see INTEGRATION keyword). This implies that the cut-off criteria are automatically made more strict if a higher numerical integration accuracy is chosen. The same is true for the other parameters in the LINEARSCALING block.

OVERLAP_INT

determines the overlap criterion for pairs of AO's in the calculation of the Fock-matrix in a block of points. Indirectly it determines what the cut-off radii for AO's should be. The value of ovint has a strong influence on the timing for the evaluation of the Fock matrix, which is very important for the overall timings. The default value for ovint is accint + 2 (typically 6). Again, a higher value implies a safer but slower calculation.

PROGCONV

determines how the overall accuracy changes during the SCF procedure ('progressive convergence'). The idea is that one might get away with a lower accuracy during the initial SCF cycles, as long as the last cycle(s) is/are sufficiently accurate. The current default is that progconv has the value 0, which means that the accuracy in the beginning of the SCF is the same as in the rest of the SCF. This keyword is currently still in the testing phase, so we do not recommend changing its default value. The value of progconv determines how much lower the other parameters in the LINEARSCALING input block are at the beginning of the SCF than at the end.

CUTOFF_COULOMB

determines the radii for the fit functions in the evaluation of the (short-range part of) the Coulomb potential. As the Coulomb potential may take a sizable amount of time, the value chosen for epsvc may influence the total ADF timing significantly as well. The default value for epsvc is accint + 4 (typically 8).

CUTOFF_MULTIPOLES

determines the cut-offs in the multipole (long-range) part of the Coulomb potential. This term scales quadratically with system size, but has a small prefactor. In most cases, change in the epsmp value will not affect the CPU time significantly. The default value for epsmp is accint + 4 (typically 8).

All Points

ADF makes use of symmetry in the numerical integrations. Points are generated for the *irreducible wedge,* a symmetry unique sub region of space. Optionally the symmetry equivalent points are also used. This is achieved by setting the key

```
ALLPOINTS
```

The key has no argument. The CPU time increases roughly by a factor equal to the number of symmetry operators, and the results should be the same. This key is available only as a debugging feature, to check the correctness of certain symmetry related algorithms.

Full SCF

During a geometry optimization the SCF convergence criterion is relaxed as long as the geometry has not yet converged. The value actually in effect depends on the current maximum (Cartesian) gradient as printed in the geometry update section. The effective convergence criterion is kept between the primary (final) and secondary criterion respectively which are both controlled by the key SCF (subkey *convergence*). The key FULLSCF turns this feature off: the primary criterion will apply always.

```
FULLSCF
```

The same effect is achieved by specifying the secondary criterion (key SCF) to be the same as the primary one.

Full Fock

At every cycle in the SCF procedure the Fock operator is computed in all integration points. By default the *difference* with the values of the previous cycle are used to compute *changes* in the Fock matrix elements. This leads in general to better computational efficiency in two ways: 1) when all such difference values in a block of integration points are very small such a block is skipped in the calculation. 2) if the values are not negligible but still rather small, the contribution from such a block to matrix elements between basis functions with small overlaps are neglected.

With the key

```
FULLFOCK
```

this is turned off, so that the complete matrix elements are computed, no blocks are skipped and the neglect of matrix elements between functions with small overlaps (see also the key TAILS) is controlled solely by the function characteristics and precision requirements, not by the development of the SCF.

Electrostatic interactions from Fit density

By default the program tries to evaluate the electrostatic Coulomb interaction energy between the fragments in a molecule using the exact fragment charge densities. The implemented algorithm requires that all fragments are spherically symmetric. This is checked by the program by verifying that all fragments have been computed in atom symmetry. It that is not the case, an alternative method is applied, using the fitted charge densities of the atoms; this is an approximation with a small, but not insignificant error. The following key forces the program to apply the fit density approach even in the case of spherically symmetric fragments. This aspect applies only to the final bonding energy analysis, not to energy computations and their gradients within the automatic geometry optimizer. The purpose of this option is to simulate a previously existing situation where the electrostatic term in the bonding energy was computed from the fit density regardless of the fragments and their internal symmetries.

```
FITELSTAT
```

presence of this key in the input file triggers using the fit density.

Save info

Several types of information, gathered during the run, are lost on exit. The SAVE key allows you to prevent the removal of such information.

```
SAVE info
```

`info`

A sequence of names separated by blanks or commas. save may occur any number of times in the input file.

save can also be used in the negative form:

```
NOSAVE info
```

The structure is similar:

info is a list of arguments and nosave may, like save, occur any number of times in the input file.

save and nosave turn save-info options on and off. A lists of the available options, with their default status.

item	*default*	*explanation*
TAPE10	no	File with numerical integration data: points and weights, values of functions (depends on direct-SCF options) and core densities and potentials.
TAPE11	no	File with fit integrals.
TAPE13	no	Check point file. This file is lost (by default) only upon normal program exit, i.e. a program-controlled termination (including a program-detected error condition leading to controlled exit). In all such cases all info on TAPE13 is also present on TAPE21. tape13 exists when the program crashes into a core dump for instance, in which case it is uncertain what the contents of TAPE21 will be. The save feature allows you to specify that TAPE13 is kept *also* upon normal exit.
TAPE14	no	Scratch file with numerical integration data, mainly pertaining to individual fragments.
Timing	no	During an adf calculation the program gathers a large amount of timing information about the performance of different program parts. It can be printed, at various levels of detail, on standard output (key PRINT). It can also be stored on TAPE21, for later inspection, in a section Timing.

Table VII. Arguments for the keys save and nosave.

3 Recommendations, problems, Questions

3.1 Recommendations

Precision

The quality of the calculation, *given* the selected model Hamiltonian - density-functional, relativistic features, spin-restricted/unrestricted... - is determined to a large extent by several technical precision parameters.

The most significant ones are:

`Basis set`

Obviously, the quality of the basis set may have a large impact on the results. As a general rule, minimum and almost-minimum basis sets (types SZ and DZ, old names I and II) may be used for pilot calculations, but polarization functions should be included (DZP, TZP, old names III, IV) for more reliable results.

`SCF convergence`

The self-consistent-field (SCF) and geometry optimization procedures terminate when convergence criteria are satisfied. If these are set sloppy the results may carry large *error bars*. The default SCF convergence tolerance is tight enough to trust the results from that aspect. However, when the SCF procedure encounters severe problems an earlier abort may occur, namely if a secondary (less stringent) criterion has been satisfied (see the key SCF). Although this still implies a reasonable convergence, one should be aware that for instance the energy may be off by a few milli hartree (order of magnitude, may depend quite a bit on the molecule). It is recommended that in such cases you try to overcome the SCF problems in a secondary calculation, by whatever methods and tricks you can come up with, rather than simply accept the first outcomes.
Note: in a geometry optimization the SCF convergence criteria are relaxed as long as the geometry optimization has not yet converged. This should generally not affect the final results: the SCF density and hence the energy gradients may be somewhat inaccurate at the intermediate geometries, but since these are not a goal in themselves the only concern is whether this might inhibit convergence to the correct final geometry. Our experiences so far indicate that the implemented procedure is reliable in this aspect.

`Geometry convergence`

This is a far more troublesome issue. Three different types of convergence criteria are monitored: energy, gradients and coordinates. The energy does not play a critical role. Usually the energy has converged well in advance of the other items. The coordinates are usually what one is interested in. However, the program-estimated uncertainty in the coordinates depends on the Hessian, which is not computed exactly but estimated from the gradients that are computed in the various trial geometries. Although this estimated Hessian is usually good enough to guide the optimization to the minimum - or transition state, as the case may be - it is by far not accurate enough to give a reasonable estimate of force constants, frequencies, and as a consequence, neither of the uncertainties in the coordinates. An aspect adding to the discrepancy between the Hessian-derived coordinate-errors and the true deviations of the coordinates from the minimum-energy geometry is that the true energy surface is not purely quadratic and using the Hessian neglects all higher order terms. The gradients provide a better criterion for convergence of the minimizer and therefore it is recommended to tighten the criterion on the gradients, rather than anything else, when stricter convergence than the default is required.
The default convergence criteria, in particular for the gradients, are usually more than adequate to get a fair estimate of the minimum energy. Tighter convergence should only be demanded to get more reliable coordinate values (and in particular, when the equilibrium geometry needs to be determined as a preliminary for a Frequencies run).

`Numerical integration accuracy`

The key INTEGRATION determines the numerical precision of integrals that are evaluated in ADF by numerical integrals, primarily the Fock matrix elements and most of the terms in the gradients. In addition the integration settings also determine several other computational parameters. The demands on numerical integration precision depend quite a bit on the type of application. The SCF convergence seems to suffer hardly from limited integration precision, but geometry convergence does, especially when tight convergence is required and also in transition state searches, which are generally more sensitive to the quality of the computed energy gradients. An extreme case is the computation of frequencies, since they depend on differences in gradients of almost-equal geometries. Frequency calculations on molecules with sloppy modes suggest that a NumInt precision value of 6.0 may be required. We recommend at least 5.0 in TS searches and in tricky optimizations, and 4.0 in normal optimizations. For optimizations with user-set convergence criteria we recommend to set the integration precision at least 1.5 higher than the requested level of convergence for the gradient. For examples, a convergence threshold of 1e-3, should be combined with integration=(3+1.5)=4.5
Note: a large integration value implies that a lot more points will be used in the numerical integrals, thereby increasing the computational effort (roughly linear in the number of points). However, in optimizations and TS searches, the program will internally reduce the integration settings as long as the geometry is far from convergence, so the costs in intermediate geometry steps may not so large. See the key INTEGRATION.

Electronic Configuration

Not specifying occupation numbers in input will *not* automatically result in the computational of the ground state. It may even lead to non-convergence in the SCF and/or in the determination of minimum-energy geometries or transition states. Therefore: whenever possible, specify occupation numbers explicitly in input (key OCCUPATIONS)!

Misunderstanding results of a calculation may easily result from a lack of awareness of how adf treats the electronic configuration, which orbitals are occupied and which are empty. Unless you specify occupation numbers in input they will be determined from the aufbau principle but only during the first few SCF cycles. Thereafter the distribution of electrons over the different symmetry representations is frozen (see the key OCCUPATIONS, options AUFBAU and aufbau2). If at that point the potential has not yet sufficiently relaxed to self-consistency the *final* situation may be non-aufbau.

A related aspect is that the *ground state* does not necessarily *have* an aufbau occupation scheme. In principle, different competing electronic states have to be evaluated to determine which has the lowest total (strongest bonding) energy.

Check output always carefully as to which orbitals are occupied. In general, whenever possible, supply occupation numbers in input. Be aware that the automatic choice by the program may in a Geometry Optimization result in different configurations in successive geometries: the automatic assessment by the program will be carried out anew in each SCF procedure. If competing configurations with comparable energies have different equilibrium geometries, the geometry optimization has a high failure probability. The gradients computed from the SCF solution of a particular configuration drive the atoms in a certain direction, but in the next geometry, when the program re-determines the occupations and finds a different configuration, the resulting gradients may drive the atoms in another direction.

See the keys CHARGE and OCCUPATIONS for user-control of occupation numbers.

Spin-unrestricted versus spin-restricted, Spin states

If your molecule has unpaired electrons, you should run an unrestricted calculation, in principle. However, if this exhibits convergence problems (or if you simply want to save time: an unrestricted calculation takes a

factor 2 more CPU time and data storage), you may consider to do it in two steps. First, run a spin-*restricted* calculation. Then perform a spin-unrestricted calculation using the restricted TAPE21 as a restart file. In the follow-up calculation you should specify the precise occupation numbers for the state you're interested in, *and* use the SCF input key to specify *only one* SCF cycle (iterations=1). This prohibits convergence (so you keep the converged *restricted* orbitals) and gives you a fairly adequate approximation to a converged unrestricted result. See also the H2 example run for a discussion in the Examples document.

An unrestricted calculation does not necessarily yield the multiplet configuration (triple, doublet ...). This is a rather complicated matter, see the discussion in the Theory document.

Geometry Optimization

Bond angles of zero or 180 degrees

Avoid bond angles of 0 or 180 degrees. Use a dummy atom at a location orthogonal to the co-linear triple and define angles w.r.t. the dummy atom.

Be aware that bond angles can be explicit - these are easily recognized - but also implicit, in the definition of dihedral angles: it is absolutely imperative that such implicit bond angles are never 0 or 180 degrees: the dihedral angle will not be properly defined and an error will occur.

The program may in some cases be able to recover from 0/180 degree bond angles, but this is not a certainty. If it fails, the geometry update steps may go completely wild. Even worse: the steps may remain small but convergence is not reached, without a clear and explicit indication in the output about the cause.

Sloppy modes

Many molecules have sloppy modes, implying that geometric departures along these modes from the true minimum hardly change the energy and do not result in sizeable gradients. This usually shows up in slow convergence: energy and gradients appear to be converged but the computed step lengths, an assessment of the error in the geometry itself, do not disappear.

Starting from ADF2005.01 delocalized coordinates can be used in geometry optimizations and transition state searches. The use of delocalized coordinates often help in convergence of these problematic sloppy modes.

It depends then on the purpose of the run whether a continued search for the minimum is useful if one has slow convergence: not if you only want to know the energy at the minimum, but certainly so if you want to determine all geometric parameters to high precision. Depending on the case you may therefore want to relax the convergence criterion on the coordinate steps. In the case of Z-matrix optimization this has to be done primarily for the *angular* coordinates because the bond lengths are usually much stiffer and will therefore not suffer from sloppy mode problems. If you insist on strict convergence of sloppy modes you should use a fair integration precision (at least 4.0, preferably 5.0).

Step convergence

The criterion on convergence of the coordinates (steps) is often *not* a reliable measure for the precision of the final coordinates, although it does give a reasonable estimate (order of magnitude). To get accurate results you should tighten the criterion for the gradients, rather than for the steps.

Basis Sets for Organic Molecules: Single-zeta vs. Double-zeta

A few tests have been done on small (less than 10 atoms) and medium sized (20-30 atoms) *organic* molecules (not containing transition metals) to compare double-zeta with single-zeta (minimal) basis sets. The two procedures were: *a*) a straightforward geometry optimization with the double-zeta basis set, and *b*) a geometry optimization with a single zeta basis, followed by a double zeta *single-point* calculation in the optimized geometry to evaluate the bonding energy and other properties. The results differed very little as regards the final geometry and therefore also as regards the energy etc.: from less than 0.01eV for small molecules to 0.25eV for a 26-atom case (debrisoquine CHN).

The additional single-point double-zeta calculation, required to obtain the bonding energy, makes that the computational costs are not automatically lower for the single-zeta optimization procedure. For the smaller molecules they take indeed some 25% *more* time. For the larger molecules this seems to get reversed however, the single-zeta approach being less than half as expensive as the double zeta.

Results for optimized bond-lengths in single-zeta bases are found to be very *in*accurate when Sulphur atoms occur in *chains*. In such molecules these atoms need a 3d-polarization function, as provided in adf's standard type-DZP (old name III) (double-zeta plus polarization) basis sets. Results so far suggest that this particular problem with single-zeta bases does not occur when such atoms occur in a *ring* rather than in a *chain* within the molecule. Probably this is related to the empty d-shell in the atom being rather close to the valence p-shell. It can be expected that the same behavior will be displayed by Phosporus. Obviously one should not rely on such generalizations too strongly; it is sensible to always run a few tests and verify whether it can be applied to the case at hand.

Although the remarks above suggest a promising time-saving approach for the optimization of larger organic molecules, the conclusions must be considered with very great care since these investigations have been carried out only for a very small number of molecules. Also one should be *extremely* careful to extend the conclusions to transition metal complexes.

Frequencies

Outcomes of Frequencies calculations are usually quite sensitive to the geometry, so before computing the frequencies, one should make sure that the geometry is well converged *at the level of the subsequent Frequencies calculation:* the same model parameters and basis sets.

In all cases one should take care that the precision of Numerical Integration is adequate, preferably *at least* 5.0 (this is good advice anyway for a sound Frequencies calculation).

Doing one-point, rather than two-point differentiation will roughly save you half of the time needed to complete the calculation. Increasing the integration precision will work the other way. To obtain high-precision results using one-point differentiation requires for one thing that you use very small displacements (smaller than the defaults) *and* high accuracy of numerical integration. Recent studies [14, 108] suggest to use *a*) two-sided displacements, *b*) an integration precision of 6.0 (!!).

This may not always be feasible due to the high CPU costs, but it should at least stress the importance of accuracy in the computation of frequencies.

A computation of frequencies runs over discrete displacements of atomic coordinates. When using Cartesian displacement coordinates, the program applies symmetry to skip symmetry-equivalent displacements and thereby save CPU time. In the output and logfile you'll find in such a case that the 'frequency displacement counter' skips one or more values: the counter counts all possible displacements, while only the symmetry-unique ones are actually carried out.

Starting from ADF2005.01 symmetric displacements can be used. This speeds up the computation significantly and reduces the level of numerical noise in gradients by using the SMOOTH option.

Relativistic methods

The ZORA relativistic approach is often superior and in other cases at least similar to the older Pauli method. In particular for all-electron calculations generally, and for very heavy elements even within the frozen core approach, the Pauli method may exhibit significant shortcomings. This is mostly due to the variational instability of the Pauli formalism in the deep-core region near the nucleus. The bigger the basis set and the smaller the frozen core, the more likely this will show up, while generally speaking you might be tempted to use smaller cores and bigger basis sets to *improve* your results. The ZORA approach does not suffer from these problems and is, therefore, highly recommended over the Pauli formalism.

3.2 Trouble Shooting

This chapter contains hints to help you solve some problems and comments on frequently asked questions.

License file corrupt

You may find that, after having installed the license file, the program still doesn't run and prints a message like 'your license file is corrupt'. To explain how this may come about, and how you overcome this, a few words on license files.

Each license file consists of pairs of lines. The first of each pair is text that states, in more or less readable format typical aspects such as an expiration date, the version number of the software and so on. The second line contains the same information in encrypted format: a (long) string of characters that seem to make little sense. The program reads the license file and checks, with its internal encrypting formulas, that the two lines match. If not, it stops and prints the 'corrupt' message. So, there are two common reasons why it may happen to you:

- You are using a license file for another version of the software than your executables correspond to. Newer (major) releases may contain a different encrypting formula, so that the match in old license files is not recognized anymore.
  So, please verify that your license file and executable belong to the same major release.
- More likely: the license file as it has been created has been modified in some way.
  Sometimes, people inspect it and 'clean it up' a little bit, for instance by removing 'redundant' spaces, or by making some other 'improvements'.
  Unfortunately, every such modification will destroy the encryption match and lead to the 'corrupt' error.
  Most of the times, however, the reason lies in the mailing system, by which the license file has been sent
  to you. If the encrypted line is rather long, the mailer may have cut it in two shorter lines.
  To verify (and correct) this: edit the license file and see if it consists of pairs of lines as described above.
  If not, re-unify the broken lines and try again.
- Finally, the problem may lie in your O/S, which may have inserted additional hidden <CR> characters (Carriage-Return) into the license file. You can remove them with our fix_license utility (in $ADHFOME/Install), see the Installation manual.

Recover from Crash

A calculation may terminate in two ways: controlled or uncontrolled. Controlled termination includes cases where the program itself detects an error and decides that continuation of the calculation is impossible or pointless. In all such cases the standard exit routine is executed, resulting in an output section with some final information. This also ensures that the general result file TAPE21 is closed properly and all relevant information flushed to it.

Uncontrolled termination may occur, for instance when some bug causes the program to divide by zero, violate memory access restrictions, etc. Usually this leads to an immediate abort of the program by the Operating System and hence loss of control by the program. In such situations the information on TAPE21 may be incomplete because some of the data are kept in memory until the final termination of the program is carried out. It would be a terrible nuisance to see all time spent so far being lost. To remedy this adf supports a check point file, named TAPE13, to help you recover at least some, if not most, of the results: not for analysis, but for continuation from a point not too long before the fatal condition occurred. TAPE13 can be used, just like TAPE21, as a normal restart file. See the restart key.

Memory Management

Insufficient Space for Allocation

Problem: The program aborts with error message insufficient space for allocation. This message is issued both in the logfile and in the output file.

Cause: The total amount of memory available for dynamic allocation of arrays is exceeded.

Cure: You can decrease the amount of memory needed with the key VECTORLENGTH. Using more processors may help to reduce the amount of memory needed. Sometimes calculations are just too big to fit on your computer.

Note: If workspace problems occur for relatively small calculations, there might be a bug. Notify your adf contact: send us the output file so that we can have a look and check things out.

SCF

No convergence

Problem: The SCF procedure does not converge, the errors of successive cycles don't diminish. This may be due to any of a large number of causes. Some of the more frequently occurring:

Possible Cause 1: Electrons are hopping back and forth between two (or more) spatially very different orbitals at successive cycles. The reason that this may happen is that in dft virtual (empty) one-electron orbitals tend to have a too-low orbital energy compared to the occupied ones, so that, if the true HOMO-LUMO gap is small enough, the computed spectrum may produce an empty orbital *below* an occupied one. Consequently, on the next SCF cycle the program may assign electrons according to the aufbau principle (say) and hence put electrons in orbitals that were empty on the previous cycle, and vice versa. On the next cycle, however, the now-occupied orbital gets a higher energy and the energy ordering is reversed again, resulting in re-adjusting the assignment of electrons. This may lead to strong oscillations and non-convergence.

Cure: Apply the feature keeporbitals (key OCCUPATIONS), to let the program try to keep orbitals occupied that are spatially similar over subsequent cycles, rather than assigning electrons according to energy ordering.
Often one can converge a system in which one equally occupies the problematic orbitals in a spin-restricted calculation. One can then use the TAPE21 from this calculation as a starting point for the next calculation with the keeporbitals feature (key OCCUPATIONS).
You may also apply strong damping (option mix, key SCF), and suppress the DIIS procedure (key SCF, option DIIS, specifications ok and cyc): all these tricks may help to let the program operate more carefully (and more slowly) in SCF updating. Note that the default SCF cycle at which the DIIS will start is rather low. This value can be increases with the specification cyc in the option DIIS of the key SCF.
For very problematic cases one can use the option smearq of the key OCCUPATIONS to smear the electrons out over orbitals that lie around the fermi level and that are close in energy. One can then use the TAPE21 from this calculation as a starting point for the next calculation with a lower vale of the smearq parameter, etecetera. Note that this may introduce fractional occupation numbers.

Possible Cause 2: You have put a wrong number of (valence) electrons in the system, so that you are, for example, trying to compute a highly ionized system. Although this should in principle not always give rise to SCF problems, it turns out to happen often in practice.

Cure: Check that the net total charge of the molecule is correct.

If you want to compute a strongly charged system that seems to cause SCF problems, try to use the Keeporbitals feature (key OCCUPATIONS). See also previous cure.

Remark: In case of SCF convergence problems, *always* check carefully whether it is a *technical* problem or a *physical* one. Technical problems may be addressed with various SCF strategy parameters.
More often you'll find it is in fact a *physical* problem. The system at hand may have two or more configurations that are competitive in energy, so that a one-determinant wave function approach is not suitable anyway, even within DFT. In such a case you should reconsider what you want in fact to be computed.

Convergence difficulties with spin-unrestricted calculations

If spin-unrestricted calculations fail to converge, you may try to run first a restricted calculation. Then perform the unrestricted while using the TAPE21 from the earlier run as a restart file. Now, in the restart run you have two alternatives that may stand a better chance to give you the desired result than your original failing calculation:

(1) Apply the MODIFYSTARTPOTENTIAL key to steer the (at the least the initial) spin-density on a per-atom (fragment) basis towards what you expect should be the final self-consistent situation. It may be necessary to apply stronger damping, or do other SCF convergence tricks like level shifting, in order to preserve your initial construction long enough for the system to get near the minimum it should converge to. It is recommended to supply occupation numbers (in case the system has symmetry) in input.

(2) If all SCF tricks fail (but you did get the *restricted* converged result), you may simply run a one-cycle unrestricted run, specifying the occupation numbers you consider appropriate and so obtain the unrestricted occupation state for the restricted-self-consistent MOs. The difference with the 'true' converged *un*restricted MOs is often quite small. See the Examples document for the discussion of a similar case (H2).

Convergence difficulties with solvation calculations

If you are using COSMO, and experience problems with SCF convergence, one additional cause of these problems can be the accuracy of solving the COSMO equations. You can increase the accuracy of solving the COSMO equations with (see also the SOLVATION kewyord):

```
SOLVATION
  ..
  Charged conv=1e-10 iter=1000
END
```

Geometry Optimization

No convergence

Problem: In a Geometry Optimization there is no progress: the atomic positions hardly change or oscillate around, while the energy *gradients* don't go to zero.

Possible Cause: Occupation numbers have not been supplied in input (OCCUPATIONS). For some molecules the procedure gets stuck between two (or more) different electronic configurations, with gradients pointing in different directions for the competing states. In each new SCF procedure, after a geometry update, the occupation numbers are re-determined, by default according to the aufbau principle. As a consequence, the successive SCF procedures may handle different electronic configurations and hence produce contradicting

geometry-updates. See the key OCCUPATIONS.

Check: in the output file the occupation numbers that have been in effect during the successive SCF procedures. If they are different, then:

Cure: supply occupation numbers in input.

Alternative cause: SCF convergence not reached, or the criterion was too weak, or the precision of numerical integration in insufficient. Such causes may lead to inconsistencies between the true energy surface properties and the computed gradients. Usually this will only slow down the convergence but not prohibit it. However, if the SCF convergence is really problematic, it might get more serious. Reconsider the SCF strategy parameters, the Numerical Integration precision, or try keeporbitals (a subkey to the key OCCUPATIONS).

Notes: The accuracy of the gradients can be made higher if the DISHUL parameter in the INTEGRATION block key is increased, for example, to dishul=5. One can use the same integration and convergence criteria for successive cycles, for example: `INTEGRATION 5 5`, and in the SCF key use the option: `converge 1e-6 1e-6`. The errors then can become more systematic (instead of random).

Spurious jumps

Problem: during geometry optimization, the atomic configuration makes a large and unrealistic jump.

Possible cause 1: the triplet of atoms to which the current atom is related by the Z-matrix is (almost) co-linear. When, in a geometry step, the triplet passes through co-linearity, the dihedral angle for the current atom should make a discontinuous jump of 180 degrees. This is not checked in the program and the dihedral angle may not get corrected, resulting in a geometric jump of the atom (and hence of all atoms related to it by the Z-matrix).

Check: the triplets of atoms, used in your Z-matrix to define the dihedral angles. If one of them is almost colinear, then:

Cure: redefine the Z-matrix or use *Cartesian* optimization.

Alternative cause 2: the connectivity of the Z-matrix does not reflect the important bonds. Especially when the molecule contains (a large number of) rings, this badly affects the stability of the geometry update step. The reason is basically that computed *Cartesian* forces are transformed into changes of the curvilinear *internal* coordinates. The transformation between the two systems of coordinates is non-linear, but mathematically assumed to be linear. This is only a good approximation for small steps.

Cure: redefine the Z-matrix and/or (if the geometry steps are very large) set a smaller upper bound on the maximum step (key GEOMETRY, subkey *step*).

Constraints are violated

Problem: constraints are violated: coordinates that were specified as frozen turn out to change during the optimization or coordinates that should remain the same start to differ after a few geometry update cycles.

Possible cause: there is an internal conflict between different demands, usually: symmetry versus constraints. The problem arises easily when a constrained optimization is requested for a molecule with some symmetry while the coordinates were defined with a Z-matrix structure that does not properly reflect the symmetry. Usually the deviations from the requested constraints are small. If they are really large, there might be a bug and you should contact an adf representative.

Cure: redefine the Z-matrix and/or use Cartesian optimization (if the constraints are expressible in

Cartesians).

Clearly wrong results (bond lengths)

If the computed equilibrium geometry appears to exhibit unlikely values, typically significantly *too short* bond lengths, you may have run into a basis set problem, in particular (but not only) if the Pauli relativistic method is applied.

Problem: Optimized bond lengths are clearly too short. The energy may also look suspicious.

Possible cause 1: Basis set trouble: onset of Pauli variational collapse, if you have applied the Pauli relativistic option. Caused by small (or absent) frozen cores and/or relatively large basis sets, applied to heavy elements.

Possible cause 2: Basis set trouble also, but quite different from the previous potential cause: you have used relatively *large* frozen cores. When the atoms approach each other during the optimization and the frozen cores start to overlap, the energy computation and the computed energy gradients become more and more incorrect. This is a result of the inappropriateness of the frozen core approximation, which indeed assumes that frozen cores of neighboring atoms do not significantly overlap. Without going into a detailed explanation here, the net effect is that certain repulsive terms in the energy computation are missing and hence a spurious tendency to a 'core collapse' arises, yielding too short bond lengths.

Cure: Best is to abandon the Pauli method and use the ZORA approach instead for any relativistic calculation. If for whatever reason you insist on using the Pauli formalism, apply bigger frozen cores and, if that doesn't help, reduce the basis set (not by deleting polarization functions, but by reducing the flexibility of the occupied-atomic-orbitals space, in particular *s*- and *p*-functions). Note, however, that large frozen cores can be a cause for trouble by themselves, irrespective of any relativistic feature. If you have reason to believe that your frozen cores might be too *large*, given the resulting bond lengths in your calculation, you have to pick smaller cores (and hence be very wary of using the Pauli formalism for any relativity).

Frequencies

Imaginary Frequencies

Problem: totally unexpected significant imaginary frequencies are obtained (in a Frequencies run) where you are pretty convinced that all frequencies should be real.

Possible cause 1: problems with the electronic configuration. If there are competing configurations, the electronic *states* in the different displaced geometries may be different, resulting in energies and gradients belonging to different potential energy surfaces to be compared and combined into force constants (frequencies).

Check: orbital occupations and SCF convergence behavior: if the SCFs in the displaced geometries start with large errors and/or converge very slowly you are likely to have stumbled into different configurations, so that the results from the displaced geometries are incompatible.

Cure: This is a difficult situation that may require some experimenting and judicious manipulation of the various SCF options. The bottom line is that you should try anything you can to ensure that all involved geometries have the same electronic configuration. As long as you fail to achieve this, the results are meaningless.

Possible cause 2: flat potential energy surface (think about almost free rotation modes) coupled with relatively high noise level in gradients caused by numerical integration errors or not sufficiently converged

geometry optimization.

Check: visualize the imaginary frequencies in ADFspectra and check that their respective normal modes correspond to movements that are expected to be have (nearly) flat energy profile.

Cure:

- restart geometry optimization with more strict convergence criteria. The default criterion on gradients 0.01 Hartree/Angstrom may be not strict enough for some systems.
  In such cases a value of 0.0001 is recommended.
- use DISHUL option of the INTEGRATION keyword to a higher value, for example 5. A higher value makes
  gradients smoothing more efficient. See also the SMOOTH subkeyword

Geometry-displacement numbers in the logfile are not contiguous

Problem: successive displaced geometries in the logfile are numbered, but in your case these numbers make sudden jumps, like '0, 1, 2, 5, 6, 13...'

Cause: you're using Cartesian displacements in a system that has some symmetry in its equilibrium geometry. The program skips the displacements of symmetry-equivalent atomic coordinates to save time. The displacement counts in the logfile do not run over the actually performed displacements but over all atomic coordinates that could be displaced if no use were made of symmetry properties.

Cure: there is no error, don't worry.

Input ignored

Problem: the program doesn't get past input and aborts with a message eof while reading (....). Or the program seems to ignore some parts of input and as a consequence goes wrong somewhere. Or it seems that part of the input has not been read correctly or not at all.

Cause 1. You have used tab characters in your input file. These are not normally visible when you edit your file, but they will affect the program's scanning of the input. When you use tab characters in the input, it is very likely that the program will do something wrong somewhere. Tabs may be ignored by the program, so that items that you believed were separate (by a tab!) are in fact read as contiguous.

Check: the input file on tab characters.

Cause 2: misusage of one of the block-type keys or general keys.

A case that relatively often shows up is typing a title as first line of the input file, *without preceding it by the keyword title*. The program does not understand this as the title, but rather tries to interpret the first word as a keyword. This leads to an error if the first word is recognized as one of the pre-defined block-type keys (possibly abbreviated).

Check: the input file on usage of block-type keys and on proper usage of a title.

Cause 3: incorrect processing of expressions or unintended replacement of names by numerical values. Various kinds of mis-typing or incorrect usage of variables may cause this.

Check: how the program sees input, *after parsing*. This can be done by rerunning the job, with as first line in input : print parser.

This will cause the program to copy each input line *twice* to output, the second time after having parsed it.

You may use StopAfter Input or StopAfter Init to let the program quit early so you can inspect what is going on with the input reading.

SFO Populations

In the section that prints the SFO populations of (selected) MOs you may occasionally find, for some SFOs in some MOs, *negative* SFO contributions. This may seem unphysical and hence suspicious, but it is 'only' a result of the Mulliken-type analysis method that underlies the computation of the SFO contributions. See the section below that discusses the output file. Likewise for larger-than-100% contributions: don't worry too much, these numbers may be correct (mathematically, given the Mulliken population formulas).

Error Aborts

The program performs a large number of checks during the calculation and may stop when it detects and error. It is close to impossible to show here a complete list of all possible error messages. In a large number of cases, additional information is printed in the output file to provide a clue as to the cause of the error. It is always useful to carefully inspect the printed info and to try to understand the meaning of any error- or warning messages. If you can't find your way out, try to get help from your adf provider. If that fails, contact us directly at support@scm.com

Warnings

The program attempts to detect bugs, instabilities, convergence problems, et cetera and may issue warnings when something looks suspicious. This is not necessarily fatal to your results, but you should be cautious and try to understand what the messages are about. Most warnings are printed in the logfile. Usually there is corresponding and more extensive information in the standard output file.

3.3 Questions

`Overlap matrix in BAS representation`

How do I get the overlap (S) matrix in the BAS representation?
It is stored on a scratch file TAPE15, which is normally deleted at the end of the calculation because it can be pretty big. To retain that file, insert 'SAVE TAPE15' in your input, see the Save key.
TAPE15 is a KF file, which you can manipulate with the KF utilities. On TAPE15 the overlap matrix is stored as the variable 'smat' in the section 'Matrices', in reduced (triangular) format: (1,1), (1,2), (2,2), (1,3) et cetera

4 FILES

ADF produces two ASCII files: standard output and the log file. The latter is a very concise summary of the calculation's progress during the run. Furthermore, adf produces and reads binary data files. Most of these files have the so-called KF format. KF stands for Keyed File: KF files are keyword oriented, which makes them easy to process by simple procedures. KF files are Direct Access binary files. Consult the utilities document for how to use some standard utilities for processing kf files.

4.1 Parallel Execution

If a parallel version of ADF has been installed *and* the installation parameter hasowndirectory in the file settings is false (see the Installation manual), then all parallel processes generate and use files in the same directory. In this situation the different kids must be able to discriminate between their own files and those of other kids. This is achieved by modifying the filenames: an integer is appended with an underscore. So, rather than TAPE10 (say), we would have tape10_0 for the parent process and tape10_1, tape10_2, etc. for the kids. This does not need to bother you if you use the program run scripts provided with the package: these control execution of adf in parallel (pvm). See the Installation manual and the Examples document for more details.

4.2 Standard output

The (standard) output file contains information of the main characteristics of the run, the SCF and geometry optimization results, bonding energy and population analyses. Major parts of output can be regulated with print switches, see the keys (no)print and eprint.

By default the program produces quite a bit of output, for a large part related to (Mulliken-type) population analyses of the molecule in total, as well as of individual orbitals, both in terms of the elementary basis functions and in terms of the SFOs, the symmetry-adapted Fragment Orbitals.

The fragment-oriented approach of adf is very suitable for a thorough chemical analysis of molecular orbital properties and a conceptual representation of results. New users are advised to spend time and get familiar with the SFO-type analysis. It is an extremely more powerful tool to understand the electronic structure of the molecule than the classical atomic orbital populations.

A summary of output is given below, assuming that default values apply for all print switches. Keep one of the Example outputs at hand when reading the description below.

Input Echo, Output Header

- Copy of the input file, except any InLine records: these are expanded and the contents of the inlinefile replaces the InLine command in the echo.
- Header with the program name, the release number and a copyright statement.
- Directly below the header are printed the job identification, title, and any comments that may have been supplied via input (key COMMENT).
  The job identification is comprised of the adf release number and the date and time of the calculation.

Main Job Characteristics

- The Model Parameters such as the Density Functional and relativistic options.
- A list of attached files: restart data files and fragment files.
- The run type: Geometry Optimization, Frequencies...
- (Initial) geometric data: atomic positions, atom types, defined fragments, and the inter-atomic distance matrix.
- The point group symmetry, with a list of the irreducible representations and subspecies.
- The electronic configuration: occupation numbers (if specified), their distribution over spin-α and spin-β,
  and the net charge of the molecule.

Build Info: Fragments and Function Sets

See the print options eprint:*frag*, eprint:*sfo* and functions.

- Correspondence between fragments in the molecule and the corresponding *master* fragments on the pertaining fragment file. (This output is by default off)
- SFOs: the Symmetry combinations of Fragment Orbitals. The SFOs are the basic conceptual entities for the analysis of MOs and other results.
  Note: The FO *coefficients* that expand the SFOs are normalized in the sense that they add up (squared) to unity.
  The resulting SFO *function* is not necessarily a normalized function. The FOs are normalized, so it depends on the *overlap* between the FOs what the self-overlap and hence the norm of the SFO is.

Also printed are, for each subspecies in each irrep separately, the indices of the elementary basis functions
from which the FOs and hence the SFOs are built up. (The overlap matrix of SFOs is printed much later, in the SFO Populations section after everything (SCF, Geometry) has cycled to convergence).

- The elementary basis functions, fit functions, and the frozen-core levels of the atoms.
  First the lists of function *sets*, defined by radial behavior and the angular quantum number, are printed for all atom types on which the functions are centered. Thereafter follows the complete BAS list where the function sets have been expanded over all atoms (the *sets* are printed only for the atom *types*) and also over all Cartesian harmonics (6, not 5 *d*-functions, et cetera).
  In this printout the numbering can be found to which the SFO survey above refers.

Technical Parameters

See the PRINT key techpar.

- Parallelization and vectorization characteristics.
- *Direct* versus *Store-On-Disk* options.
- Update strategy parameters for Geometry updates (if applicable) and for the SCF procedure.
- General precision settings for numerical integration and neglect-of-small function values (in integral evaluations).

Computational Report

See the print switches computation, eprint:*numint*, eprint:*SCF*, eprint:*geo*.

`Numerical integration`

General grid-generating parameter(s) and the number of generated (symmetry unique) integration points, with their distribution over the distinct kinds of integration regions: the atomic (core-like) spheres, the remaining interstitial regions between the atoms (atomic polyhedra), and the outer region, i.e. the part of space around the molecule.

Partitioning of the points in blocks. In general there are too many integration points to have all pertaining data (values of basis functions in the points etc.) in memory. A segmentation in blocks of points is therefore applied, processing a block of data at a time after loading it from disk or recomputing it (depending on the *Direct* options). This also determines vector lengths and hence vectorization performance in numerical integral evaluations.

Integration Tests. The generation of the points involves an adaptive procedure to tune the point distribution such that a pre-set precision of several test integrals is achieved with a minimal number of points. The generated scheme is *a posteriori* tested by evaluating a few integrals in the actual molecule. This does not result in any subsequent adaptation of the grid but only produces info for the user to verify that all goes well. If the results are suspicious a warning is issued and if the results are too bad, the program will abort. The most important and significant test is the evaluation of the self-overlaps of all symmetry-adapted elementary basis functions. The maximum and root-mean-square (relative) errors are printed. The number of significant figures suggested by the rms error should roughly equal the accuracy parameter. This may not hold so well for extremely low values of the parameter (less than 1.5 say) where results become unpredictable. Likewise for very high values (greater than 6.0 say) where the adaptive procedure has not extensively been tested and hence the results might deviate more (not necessarily in the wrong direction!). This extensive testing is not carried out in Direct-SCF (bas) mode because in that case the necessary information is not available (basis functions are only computed when needed in the SCF).
A test that is always carried out is the numerical integration of the total frozen core density (summed over all atoms in the molecule). Also here a warning or even abort will occur when the result indicates that the

integral has insufficient accuracy compared with the integration precision parameter.

`SCF procedure`

at each cycle: for each irreducible representation: the one-electron orbital energies and the occupation numbers for a contiguous sequence of orbitals. The indices of the lowest and highest MOs (in energy ordering) are printed directly after the irrep label. With this information you can check the electronic configuration. When convergence is problematic, more info appears at the higher iterations.
The involved orbitals are usually the highest few occupied and the lowest few unoccupied orbitals, see the eprint subkey eigval. During the SCF, as soon as the distribution of electrons over irreps is frozen, only the occupied orbital energies are computed and hence printed.
Also printed at each SCF cycle is the difference of the density matrix (P-matrix) with the previous cycle: the average and maximum difference in the diagonal elements.

At the end of the SCF: concise information about the density-fit precision: the error integral for the SCF density. The error integral is the integral of the difference between the exact density and the fit density, squared. Such values have very little to do with numerical integration, rather they show whether or not the employed set of fit functions are adequate to describe the SCF density. Error integral values that significantly exceed 1e-4 times the number of atoms are suspicious and may indicate some deficiency in the fit set for the actual calculation.
On the last geometry (in an optimization) the fit-error integrals are also printed (in the Results section, see below) for the initial (sum-of-fragments) density and the orthogonalized fragments (see Chapter 1.2)

- Gross atomic charges, computed from a Mulliken population analysis.
- Geometry Updates. The contents of this section depends on the RunType:
  Geometry Optimization, Frequencies.... It is absent in a Create run and in a SinglePoint calculation.
- Gradients on the atoms: derivatives of the energy w.r.t. changes in the nuclear coordinates.
- Summary of convergence issues. One of the items considered for convergence is the maximum Cartesian
  gradient. This value corresponds in principle to one of the Gradients on the Atoms.
  Differences may occur due to user-set and automatic constraints.
  The printed Gradients are the raw gradients, the maximum Cartesian gradient is the maximum over *relevant* gradients: this ignores gradients in frozen coordinates.
  Furthermore, gradients in coordinates that are forced to remain equal are averaged before the maximum is selected; finally the raw gradients are processed to eliminate spurious components such as gradients in rigid motions (translations and possibly rotations).
  In a Z-matrix optimization any user-set constraints apply to the Z-matrix coordinate-derivatives and the maximum Cartesian gradient is selected from the Cartesian gradients that are recomputed from the constrained z-matrix gradients.
- New coordinates: Cartesian and z-matrix if applicable. Optionally the new inter-atomic distance matrix is given (not by default).

The Computational info is repeated in all cycles (SCF and geometry) until the iterations have terminated.

Results

See also Chapter V.

Details of the Results part in the output file depend on the run type. For output in a Frequencies run, see below. In other applications:

Nuclear and Electronic Configuration

- The final atomic coordinates (only in an optimization run).

- One-electron orbital data: occupation numbers and energies, HOMO and LUMO energies and, if applicable, a list of partially occupied MOs.
- Orbital energies of the Core Orbitals

ESR Properties

In an ESR calculation only (keywords ESR, QTENS):

When the G-tensor is computed (spin-orbit coupling, spin-unrestricted and collinear or spin-restricted and one unpaired electron)

- The eigenvectors of the G-tensor and the isotropic value
- The eigenvalues of the G-tensor.
- The effective spin used.

When the G-tensor is computed in a spin-orbit calculation, which is spin-restricted and one unpaired electron, then also:

- The MO of the unpaired electron, in BAS representation and in Lowdin representation.

Then the A-tensor and Q-tensor parts (Q-tensor if the QTENS keyword has been specified in input). For each atom:

- The isotope characteristics (nuclear spin, g-value and quadrupole moment)
- The position of the nucleus in the molecule (Cartesian coordinates)
- The eigenvectors of the A-tensor (Nuclear Magnetic Dipole Hyperfine interaction)
- The eigenvalue (if the g-value of the nucleus is non-zero, the eigenvalues have been multiplied by the g-value) and the isotropic value.
- The eigenvectors and eigenvalues of the Q-tensor.
  The eigenvalues have been multiplied by the quadrupole moment if non-zero.

Populations, Charge analysis

`Mulliken populations`

Mulliken populations are based on the elementary atomic basis functions (bas). The individual BAS populations are printed together with summaries of the populations in all basis functions with the same angular moment quantum number on the same atom.
A final summary is obtained by adding all functions on each atom, yielding the atom-atom populations. The atom-atom populations per l-value can be obtained if the key EXTENDEDPOPAN is included. The atomic gross charges are derived from the net and the overlap populations in the usual way.
In addition, a population analysis may be given of individual MOs (by default this is suppressed). See the EPrint keys SCF (option mopop) and orbpop.

`Hirshfeld charges`

Of the three methods applied in adf to compute charges (Mulliken, Hirshfeld, Voronoi) we recommend the Hirshfeld analysis [125, 126] and the analysis based on Voronoi *deformation* density (VDD) charges [109, 127], see below. The fragments to which the Hirshfeld charges apply are enumerated in the early geometry part of the output file, where for each fragment the numbers of the atoms are given that belong to the fragment.
The sum of the Hirshfeld charges may not add up to the analytical net total charge of the molecule. Any deviation from this is caused by numerical integration precision (small effect) and the neglect of long-distance terms that adf uses to speed up the integral evaluations. This approximation does not affect

very much the energy and molecular orbital properties, but it does show up in the sum-of-charges somewhat more. It does not indicate an error (unless the deviation is really large, say in the order of 1‰ of the total number of electrons).

`Voronoi Deformation Density (VDD) Charges`

The VDD method is based on the *deformation* density and a rigorous partitioning of space into non-overlapping atomic areas, the so-called Voronoi cells [109, 127, 128]. The Voronoi cell of an atom *A* is the region in space closer to nucleus *A* than to any other nucleus (cf. Wigner-Seitz cells in crystals). The VDD charge of an atom *A* monitors the *flow* of charge into, or out of the atomic Voronoi cell as a result of 'turning on' the chemical interactions between the atoms. The VDD method summarizes the three-dimensional deformation density on a per-atom basis. It is conceptually simple and affords a transparent interpretation based on the plausible notion of charge redistribution due to chemical bonding, i.e. the gain or loss of charge in well-defined geometrical compartments of space. For the use of VDD in analyses involving molecular fragments, see Ref. [129].

In the same fashion as for the Hirshfeld analysis, a summation over all atoms is given which should yield zero (for a neutral molecule). The deviation from zero is caused by numerical integration and by neglect-of-long-distance-terms; the same remarks apply as for the Hirshfeld analysis above.

`Multipole Derived Charges (MDC)`

This charge analysis uses the atomic multipoles (obtained from the fitted density) up to some level X, and reconstructs these multipoles exactly (up to level X) by distributing charges over all atoms.

`Mayer Bond orders`

The Mayer bond order between two atoms is calculated from the density and the overlap matrices (key EXTENDEDPOPAN), see Ref. [140]. See for Mayer bond orders and alternative definitions of bond orders also the key BONDORDER.

Dipole moment, Quadrupole moment, Electrostatic potential

Dipole moment. Note that in a ion the value of the dipole moment depends on the choice of the origin, as follows from elementary electrostatic theory.

Quadrupole moment. Note that the value of the quadrupole moment often depends on the choice of the origin, as follows from elementary electrostatic theory.

Electrostatic potential at the nuclei: the Coulomb potential of the molecule at the nuclear positions, where the contribution from the nucleus itself is omitted.

Energy and MO analysis

`MOs expanded in SFOs`

This gives a useful characterization of the character of the self-consistent molecular orbitals. Additional information is supplied by the SFO population analysis, see below.
The definition of the SFOs in terms of the Fragment MOs has been given in a earlier part of output (section build). The SFO occupation numbers that applied in the fragments are printed. This allows a determination of the orbital interactions represented in a MO.
Be aware that the bonding/antibonding nature of a SFO combination in a mo is determined by the relative signs of the coefficients *and* by the overlap of the SFOs. This overlap *may be negative!* Note also that SFOs are generally *not* normalized functions. The SFO overlap matrix is printed later, in the SFO-populations part

below.

`Bond Energy analysis`

The bond energy and its decomposition in conceptually useful terms: Pauli (exchange) repulsion, total steric repulsion, orbital interactions (partitioned into the contributions from the distinct irreducible representations), and corrections for some approximations (fitting and Transition State analysis procedure).
For a discussion of bonding energy decompositions and applications see e.g. [110, 112, 130-136]

`SFO population analysis`

For each irrep:

- Overlap matrix of the SFOs. Diagonal elements are not equal to 1.0 if the SFO is a linear combination of two or more Fragment Orbitals. The Fragment Orbitals themselves are normalized so the diagonal elements of the SFO overlap matrix give information about the overlap of the Fragment Orbitals that were combined to build the SFO.
- Populations on a per-fragment basis for a selected set of MOs (see EPrint, subkey *OrbPop*). This part is by default *not* printed, see EPRINT subkey *SFO*.
- SFO contributions per MO: populations for each of the selected MOs. In these data the MO occupation numbers are not included, so that also useful information about the virtual MOs is obtained. The printout is in matrix form, with the MOs as columns. In each printed matrix a row (corresponding to a particular SFO) is omitted if all populations of that SFO are very small in all of the MOs that are represented in that matrix. See eprint, subkey *orbpop*.
Note that this method to define SFO populations (for orbitals) is very similar to the classical Mulliken type analysis, in particular regarding the aspect that *gross* populations are obtained as the diagonal (*net*) populations plus half of the related off-diagonal (overlap) populations. Occasionally this may result in negative (!) values for the population of certain SFOs, or in percentages higher than 100%. If you have such results and wonder if they can be right, work out one of the offending cases by hand, using the printed SFO overlap matrix and the printed expansion of the MOs in SFOs to compute 'by hand' the population matrix of the pertaining MO. To avoid doing large calculations it is usually sufficient to take only the few largest MO expansion coefficients; this should at least qualitatively give the correct outcomes.
- Total SFO gross populations in a symmetry representation: from a summation over all MOs (not only those analyzed in the previous section of output) in the symmetry representation under consideration. In the gross populations the MO occupation numbers have been included.
- (Per spin): A full list of all MOs (combining all symmetry representations), ordered by energy, with their most significant SFO populations. Since there might be several significant SFO populations for a particular MO, and an SFO may actually be a linear combination of several (symmetry-related) Fragment Orbitals, this table could get quite extensive. In order to confine each SFO population specification to one line of output, the SFOs are indicated by the characteristics of the first term (Fragment Orbital) of its expansion in Fragment Orbitals. So, if you see the SFO given as the '2 P:x on the first Carbon fragment', it may actually refer to the symmetry combination of, for instance, 2P:x and 2P:y orbitals on the first, second and third Carbon fragments. A full definition of all SFOs in terms of the constituting Fragment Orbitals is given in an early part of the output.

Summary of LT or IRC path(s)

At the very end of the results section, a completed LT or IRC calculation will show tables of a few key properties in each point of the scanned path: atomic coordinates, energy, dipole moment, atomic charges and a few others, depending on the case. This gives you a quick survey of the computed profile.

Frequencies Results

In a Frequencies calculation the computed harmonic frequencies are printed. If a complete variation of coordinates has taken place, the program will compute the frequencies and normal modes also in terms of Symmetry Coordinates, along with the representation in the coordinates that were specified in input.

The zero-point energy is printed, computed as sum over frequencies:

$$E_0 = \sum \nu/2 \quad (4.2.1)$$

Any imaginary frequencies (printed in the output file as *negative* frequencies) are not included in the summation.

Thermodynamic properties (Heat Capacity, Entropy, Internal Energy) are printed, based on the ideal gas approximation. Electronic contributions are omitted. These are small when the energy gap with the next electronic configuration is large compared with the vibrational frequencies. For (near) degenerate configurations this assumption is incorrect.

Imaginary frequencies and very small frequencies are ignored in this calculation.

Exit Procedure

normal termination or an error message.

A list of all files that are (still) open when the exit routine is called. The program closes such files at this point.

Information about buffered I/O processing during the calculation.

A check of workspace to see whether all dynamically allocated arrays have been cleaned-up. If so the program mentions All Arrays Delocated. Otherwise there is something wrong and the situation will be summarized. If the calculation seems to have completed normally, but nevertheless workspace has been found not-clean, we would appreciate to get the complete output file because it might signal a programming error. This does not apply when you have used the stopafter feature: the program will then abort before the standard termination and usually not all workspace will have been cleaned up then.

Timing Statistics: a survey of cpu, System (I/O) and Elapsed times spend in various sections of the program.

Logfile

At the end of the calculation the log file is copied (optionally, see print) to the tail of the standard output file. The log file contains a concise summary of the run.

4.3 Log file

The log file (logfile) is generated during the calculation and flushed after (almost) each message that is sent to it by the program. Consequently, the user can inspect it and see what is going on without being delayed by potentially large system I/O buffers. Each message contains date and time of the message plus additional info.

A major part of the messages simply states the name of a procedure. Such messages are sent when the procedure is entered. During the SCF procedure, the SCF errors, which are a measure for non-self-consistency, are written at every cycle. In calculations where the geometry is changing (optimization, frequencies...) each set of new coordinates is sent to the log file (Cartesian, in angstrom and also Z-matrix, if a Z-matrix structure was provided in the input file). Other messages should be self-explanatory.

Be alert on error messages. Take them seriously: inspect the standard output carefully and try to understand what has gone wrong. Be also alert to warnings. They are not necessarily fatal but you should understand what they are about before being satisfied with the results of the calculation. Do not ignore them just because the program has not aborted: in some cases the program may not be able to determine whether or not you really want to do what appears to be wrong or suspicious. If you believe that the program displays erratic behavior, then the standard output file may contain more detailed information. Therefore, in such case save the complete standard output file, together with the logfile, in case we need these files for further analysis.

4.4 TAPE21

TAPE21 is the general result file of an adf calculation. It is a kf file: Direct-Access, binary, and keyword driven. It contains information about the calculation. You can use it as a fragment file in a subsequent calculation on a bigger molecule, where the current one may be a part, or in an analysis program.

The contents of TAPE21 is keyword-accessible and you may use the KF utilities (see the utilities document) for conversion of TAPE21 from binary to ASCII format and vice versa. This facility is also useful when you intend to use a TAPE21 result file produced on one type of hardware, for a continuation run on a quite different computer: Transform the binary file to ASCII format with the KF utilities on the first machine. Then transport the ASCII file to the other machine, and make a binary out of it again.

Another utility (*pkf*) can be used to obtain a summary of the contents of TAPE21. The output should be more or less self-documenting: all variables are listed by name, type (integer, real, ..) and size (number of array elements) and grouped in named sections.

The data on TAPE21 is organized in Sections which group together related data. Each section contains a number of variables. Each variable may be an array or a scalar and may be integer, real, logical or character type.

A complete dump of the contents of TAPE21 is obtained with *dmpkf*. The resulting ASCII file contains for all variables on the file:

- The name of the section it belongs to;
- The name of the variable itself;
- Three integers coding for the data of the variable:
  - The number of data elements reserved on the file for the variable;
  - The number of data elements actually used for the variable.
    In virtually all cases the number of *used* elements is equal to the number of *reserved* elements.
    The number of *used* elements is relevant for interpreting the data, the number of *reserved* elements
    has only relevance for the management of data on the file by kf-specific modules and utilities;
  - An integer code for the data type: 1=integer, 2=real, 3=character, 4=logical;
- The variable value(s).

A typical case of the contents of TAPE21 obtained by *dmpkf* operating on the binary TAPE21 file from an optimization run on H2O would be:

contents of TAPE21	***comment***
General	name of (first) section
file-ident	name of (first) variable in the current section (General)
6 6 3	characteristics of the data: 6 elements reserved on file for the variable, 6 data elements actually used, 3=integer code for the data type: character
TAPE21	Value of the variable fileident in the section General.
General	again: name of the section
title	name of the (second) variable
80 80 3	reserved and used number of data elements (both 80), and the data type code (3: character)
Water Geometry Optimization with Internal Coordinates	value
(etc.)	(etc.)

For a description of the various utilities that can be used to process TAPE21 see the ADF Properties and

Analysis documents.

Contents of TAPE21

Follows a survey of the sections and variables on TAPE21. Details may differ between different run types (SinglePoint, Frequencies...). Most items should be self-explanatory. Some are only significant for internal proceedings of the program and will not be explained in detail. The sections are described in an order that corresponds to the order in which they are generated and hence printed by the KF utility programs. However, the order of generation depends somewhat on the type of application, so some difference may be found when comparing to your own TAPE21 printout.

Note that variable and section names may contain spaces: these are significant.

A special section is the 'SUPERINDEX' section, which is in fact a table-of-contents that lists all the sections in the file, with technical information regarding their position on the file, the amount of data corresponding to that section and similar items. The SUPERINDEXX section is not discussed further here. See the KF documentation for more details.

Section General

General information about the calculation and the file

`fileident`

Name of the file. Here: TAPE21

`jobid`

adf release number with date and time of the calculation

`title`

Title of the calculation. This may have been set in the input file, or be internally generated. In a create run it is picked up from the Create database file (if no input value for the title key has been given).

`runtype`

The type of calculation, for instance SinglePoint or Frequencies

`nspin`

1 for a spin-restricted calculation, 2 for spin-unrestricted

`nspinf`

Similar for the fragment occupation numbers as they are used in the calculation, See the key FRAGOCCUPATIONS

`ldapot`

An integer code for the applied LDA part of the XC potential functional used in the SCF. 1 for VWN, 2 for VWN+Stoll ...

`xcparv`

X-alpha parameter value. Only relevant for the X-alpha LDA potential, meaningless if another LDA potential functional has been selected.

`ldaen`

As for ldapot: integer code for the LDA part of the Density Functional, now however pertaining to the (post-SCF) energy evaluation. Usually ldaen and ldapot are identical. See the key XC for details.

`xcpare`

As xcparv, but now for the energy evaluation.

`ggapot`

Specification (string) of the GGA part of the XC potential used in the SCF, for instance 'Becke Perdew'. If no GGA potential is applied, the string ggapot is empty.

`ggaen`

Similar for the GGA part of the XC energy evaluation

`iopcor`

Code for usage of frozen core: 1=use frozen cores, 0=pseudopotentials. Pseudopotentials are not supported anymore in ADF, so this variable must always be 1.

`electrons`

The number of valence electrons
Note that this is not necessarily the same as what may consider, chemically, as the valence space. Rather, it equals the total number of electrons in the calculation minus the electrons in the frozen core orbitals.

`unit of length`

Transformation factor between input-used geometrical units (for distances) and atomic units (bohr). If input of, say, the atomic coordinates is in Angstrom, the unit of length is approximately 1.89

`unit of angle`

Similar for angles. Internal units in the program are radians. Input (bond and dihedral angles) may be in degrees, in which case the unit of angle equals approximately 0.017

Section Geometry

Geometrical data such as number of atoms, coordinates, etc: Most variable names should be self-explanatory

`grouplabel`

Point group symmetry (string) used in the calculation, for instance O(H). This may be set in the input file.

`Geometric Symmetry`

Auto-determined ('true') symmetry (considering the nuclear frame and any external fields, but not taking into account any user-defined MO occupation numbers and hence the electronic charge distribution.

`symmetry tolerance`

Threshold for allowed deviation of input atomic coordinates from symmetry to be detected or verified.

`orient`

Affine transformation (3,4 matrix: rotation and translation) between the input coordinates and the frame in which the program processes the atoms. ADF has certain orientation requirements for all supported point group symmetries and may rotate and translate the input coordinates accordingly.

`oinver`

The inverse transformation of orient

`lrotat`

A logical flag to signal whether or not a rotation has been applied between the input frame and the internally used frame.

`nr of fragmenttypes`

The number of distinct types of fragments

`nr of dummy fragmenttypes`

Idem, but counting only dummy atom fragments. A dummy fragment, if it exists, must consist of one single (dummy) atom.

`fragmenttype`

Names (string) of the fragment types.

`fragment mass`

Sum of atomic masses in the fragment.

`fragment charge`

An array with 3 values per fragment type (nftypes,3): 1=sum of nuclear charges, 2=sum of effective nuclear charges (discounting for the frozen core shells), 3=nr of valence electrons

`fframe`

Signals whether or not special local coordinate frames are used for the atoms. Usually this is not so, in which case the variable has the value DEFAULT. fframe is an array that runs over the atoms. See the 'z=' option to the data in the ATOMS input key block.

`cum nr of fragments`

An array (0:nftyps) that gives the total number of fragments for the fragment types up to and including the indexed one. The ordering of fragments and fragment types is printed in the standard output file.

`nr of fragments`

The total number of fragments in the calculation
This equals the last element of the previous variable 'cum nr of fragments'

`nr of dummy fragments`

The total number of fragments that each consist of a single dummy atom.

`fragment mapping`

Affine transformation matrices (3,4: rotation and translation), one for each fragment in the molecule, that transform the fragment coordinates as they are on the fragment file(s), to the actual position of the fragments in the molecule.

`cum nr of atomtypes`

An array (0:fragmenttypes) that counts the number of atom types up to and including the indexed fragment type.

`nr of atomtypes`

Total number of atom types in the molecule. Must equal the last element of the 'cum nr of atomtypes' array

`nr of dummy atomtypes`

Similar, now counting only the atom types consisting of a dummy atom.

`atomtype`

Names (strings) of the atom types

`mass`

Atomic masses: array running over the atom types. Compare 'fragment mass'.

`charge`

Similar as for 'fragment charge', but now the values per atom type.

`cum nr of atoms`

An array (0:atomtypes) that counts the number of atoms up to and including the indexed atom type.

`nr of atoms`

Total number of atoms. Must equal the last element of the array 'cum nr of atoms'.

`nr of dummy atoms`

Total number of dummy atoms

`atmcrd`

Type of atomic coordinates in input: CART (Cartesian) or ZMAT (Internal).

`kmatrix InputOrder`

The connection matrix listing (and referencing) the atoms in the order as they were in the input file. This ordering aspect is significant because internally the program reorders the atoms and groups them together by atom type and fragment type. Hence it is relevant to know what ordering (input- or internal-) is assumed in data arrays.

`zaxis`

For each atom the direction of the local z-axis. Normally this is identical to the standard (0,0,1), but it may be different for analysis purposes. See the 'z=' option to the data records in the ATOMS block.

`fragment and atomtype index`

An integer array (natoms,2) that specifies for each atom the fragment and the atom type it belongs to.

`atom order index`

An integer array (natoms,2) that defines the re-ordering of atoms between the list in the input file and the internally used list (which is driven by fragment types, fragments, atom types; dummies come last). The

internally used list can be derived from the printout of the fragments, early in the standard output.

`kmatrix`

The connection matrix using the internally applied ordering of atoms

`xyz`

Cartesian coordinates of the atoms, in the internally used ordering of atoms

`xyz Inputorder`

Similar, but now for the ordering of atoms as in the input file.

`zmatrix`

Internal (Z-matrix) atomic coordinates

`zmatrix Inputorder`

Internal coordinates in the input-order of atoms

`Atomic Distances`

Inter atomic distance matrix

`ntyp`

Number of atom types, not counting dummy atoms,

`nqptr`

A cumulative counting array, very similar to 'cum nr of atoms'
Differences: it runs only over 'ntyp' atom types (not including dummy atoms) and its indexing as well as its values are shifted by one: nqptr(k) is the total number of atoms plus one, counting the atom types up to and including #(k-1)

`nnuc`

Total number of non-dummy atoms

`qtch`

Nuclear charges of the non-dummy atoms

`qeff`

Effective nuclear charges (subtracting charge for the frozen core shells) of the non-dummy atoms

`nfragm`

Total number of non-dummy fragments

`nofrag_1`

Integer array specifying for each non-dummy atom the fragment it belongs to.

`nofrag_2`

Integer array specifying for each non-dummy atom the fragment type it belongs to

`nuclab`

Names of the non-dummy atom types.

Section Fragments

(To be completed)

`FragmentFile`

Names of all used fragment files

`FragRun Ident,Title`

Job identification and title of each fragment run that is used in the current molecule

Section AtomTypes

(To be completed)

Section Properties

`AtomCharge Mulliken`

Atomic charges derived from Mulliken population analysis.

`Dipole`

Dipole moment in atomic units.

`FragmentCharge Hirshfeld`

Fragment charges derived from Hirshfeld analysis

`AtomCharge_initial Voronoi`

Atomic charges derived from Voronoi analysis for the initial (sum-of-fragments) charge density

`AtomCharge_SCF Voronoi`

Similar as the previous item, but now for the SCF density

`Electrostatic Pot. at Nuclei`

Coulomb potentials at the positions of the atoms, not including the contribution from the nucleus itself

Section Basis

Information about the (valence) basis set

`nbset`

The total number of basis 'sets', where a 'set' here means a Cartesian function set (3 for a *p*-type function, 6 for a *d*-type function, and so on), given by an entry in the 'list-of-basis-functions' in the data base file.

`nbaspt`

Cumulative number of basis sets (see previous variable, for 'set'), on a per atom type basis. Only non-dummy atoms (type) are considered. nbaspt(k) is 1+nr-of-basis sets up to, but not including atom type #k

`nqbas`

Main quantum number of each basis set. A 1s function has nqbas()=1

`lqbas`

Angular momentum quantum number of each basis set. The current implementation of ADF supports only *l, p, d,* and *f* basis functions, so the allowed lqbas values are 0, 1, 2, and 3

`alfbas`

The exponential decay parameters of the STO functions in the basis set

`basnrm`

Normalization coefficients for the basis sets

`naos`

The total number of basis functions, counting all Cartesian polynomials and all copies of the functions on the atoms of the pertaining atom type

`nbos`

The total number of Cartesian basis functions, *not* counting the copies of the functions on the different atoms of the atom type: the functions are defined per atom type and are (for nbos) counted only once. The next few variables relate to lists of basis functions that run from 1 to nbos: all the Cartesian polynomials, but counting the function only once per atom type. Essentially, this means counting all functions with distinct characteristics (apart from their geometrical center).

`nbptr`

Index array of the nbos functions, where the entries are the cumulative numbers of functions (+1) up to, but not including the atom type. The size of the array is (ntyp+1): one plus the number of (non-dummy) atom types.

`kx`

Powers of x of the nbos Cartesian STO basis functions

`ky`

Powers of y of the nbos Cartesian STO basis functions

`kz`

Powers of z of the nbos Cartesian STO basis functions

`kr`

Powers of r of the nbos Cartesian STO basis functions

`alf`

Exponential decay factors of the nbos Cartesian STO basis functions

`bnorm`

Normalization factors for the nbos Cartesian STO basis functions

`nprta`

Consider a list of all (naos) Cartesian STO basis functions, including copies of the functions on all atoms of the same atom type. Build that list by first taking all true valence functions on all atoms (loop over atom types, inner loops over atoms, inner loop over basis sets of the atom type, inner loop over Cartesian polynomials for the function set), then all auxiliary core-orthogonalization functions (similar loop structure). nprta(i) gives the index in that list of function #*i*, where *i* corresponds to a similar list of all naos functions in which the core and valence subsets are not separated.

`norde`

An array that runs over the non-dummy atom types. Each element gives the maximum of the main quantum number for all STO basis and fit functions corresponding to that atom type.

`lorde`

As norde, but lorde applies to the angular momentum quantum numbers.

Section Core

Information about frozen core orbitals and the Slater-type exponential functions used to describe them.

`nrcset`

The number of STO function sets to describe the frozen core orbitals in the calculation. The array is sized (0:llqcor,1:ntyp). llqcor is the maximum l-value in core orbitals (3), ntyp is the number of non-dummy atom types.

`nrcorb`

An array (0:llqcor, 1:ntyp) specifying the number of frozen core orbitals per *l*-value and per non-dummy atom type.

`ncset`

The total number of core expansion STO function sets, not counting copies on all atoms, and not counting the Cartesian polynomials (1 value per *p*-set, et cetera)

`ncorpt`

Index array: 1 + cumulative number of core expansion sets up to, but not including, the indexed atom type. The array runs from 1 to ntyp+1

`nqcor`

Main quantum numbers for the core expansion sets

`lqcor`

Angular momentum quantum numbers for the core expansion sets.

`alfcor`

Exponential decay factors for the core expansion sets.

`cornrm`

Normalization factors for the core expansion sets.

`ncos`

Total number of core expansion functions, counting all copies on different atoms of each atom type, and counting all Cartesian polynomials.

`nccpt`

Index array: 1 + cumulative number of core orbitals, counting all copies on different atoms and all Cartesian (sub) functions.

`ncptr`

Similar, but applying to the STO core expansion functions.

`ccor`

All core expansion coefficients, which express the core orbitals in the core expansion functions. The array stores the expansion coefficient sequence for each core orbital shell (not for each Cartesian sub function) and only one sequence per orbital per atom type (no duplication for the different atoms of the atom type).

`npos`

An index array. For each atom type: the index where its data are stored on the TAPE12 core data file. npos(k) may be zero if no data for atom type #k are available on TAPE12.

`kcos`

The total number of core expansion functions, like ncos, but now counting only the truly independent functions. For instance: 5 functions per *d*-set, while in ncos there are 6 functions per *d*-set. The *s*-type combination in the 6-membered *d*-set is in the calculation projected out and does not represent a degree of freedom.

`s`

The (kcos,kcos) overlap matrix of the core expansion functions. Note that, since the dimension is (kcos,kcos), the *s*-type combination has been eliminated, and likewise for the 3 *p*-type functions in each *f*-set.

`idfcor`

Integer that indicates whether or not the core set contains *d*- and/or *f*-type functions. 1=yes, 0=no

`nd`

Total number of *d*-type core orbital sets (not counting the Cartesian sub functions)

`nf`

Total number of *f*-type core orbital sets (not counting the Cartesian sub functions)

`ndorb`

An array running over the *d*-type core orbital sets (loop over atom types, loop over atoms, loop over core orbitals with *l*=2). It gives for each the index of the orbital (the first of the Cartesian subset) in the overall list of all core orbitals in the molecule (including the spurious *s*-type functions in the *d*-sets, and so on)

`nforb`

Similar as ndorb, but now for the *f*-type core orbitals.

`cmat`

Overlap matrix between core-orbitals (ncos, counting all Cartesian functions including the *s*-type function in each *d*-set, et cetera), and the basis functions. In the list of basis functions, all core functions (the auxiliary orthogonalization functions) come before all true valence basis functions, see array NPRTA.

Section Fit

This section stores information about the fit functions, which are used for the Coulomb potential evaluation.

`Unrestr.SumFrag`

A logical that flags whether or not the fit coefficients have been set and stored for the sum-of-fragments, but adjusted for the unrestricted fragments option (see the keys UnrestrictedFragments, ModifyStartPotential).

`coef_SumFrag`

Fit coefficients pertaining to the sum-of-fragments charge density.

`coef_SCF`

SCF fit coefficients.

`nfset`

Total number of fit function sets (not counting the Cartesian sub functions, not counting the copies of the functions on the atoms of an atom type)

`nfitpt`

Index array: 1+the total number of fit function sets up to, but not including, the indicated atom type.

`nqfit`

Main quantum numbers of the fit sets

`lqfit`

Angular momentum quantum numbers of the fit sets

`alffit`

Exponential decay factors of the STO fit sets.

`fitnmr`

Normalization factors for the STO fit sets.

`nfos`

Total number of Cartesian fit functions, not counting copies on all atoms of an atom type, but including all (for instance, 6 for a *d*-set) Cartesian sub functions.

`nfptr`

Index array: 1+ total number of Cartesian (see variable nfos) fit functions, up to but not including the indicated atom type.

`nprimf`

Total number of Cartesian ('primitive') fit functions, counting also the copies on all atoms of each atom type.

`nsfos`

The total number of fully symmetric (A1 symmetry) fit function combinations that represent the true dimension (variational freedom) of the space of fit functions in the calculation.

`na1ptr`

Index array, like nfptr, but applying to the nsfos symmetric function combinations.

`niskf`

This refers to an atom-limited symmetry combination of primitive fit functions, in the code and some documentation indicated as a 'g'. A 'g' is the specific part of a molecule-wide A1 fit function combination (see nsfos) that consists of all the terms that are centered on one particular atom. The number niskf gives the total number of such 'g' function combinations.
To clarify this, consider an A1 fit function combination in the molecule. Assume, that it consists of a specific linear combination the following functions: a p-x function on atom A, its partner p-y function, and the corresponding p-x and p-y functions on atom B. (Atoms A and B must be symmetry equivalent). In this example we have one A1 function (in the list of nsfos such functions) and two 'g''s. Each 'g' consists of a p-x and a p-y function combination on a specific atom.

`iskf`

Compound index array. It runs over the niskf 'g' fit function combinations and has 4 entries for each function (1:4,1:niskf). The meaning of the entries is as follows. #1=number of the fit set (not counting the copies of fit functions on different atoms of an atom type, and not counting the Cartesian sub functions) this 'g' belongs to. #2=index where the combination coefficients for this 'g' start in the arrays cofcom and numcom (see next). #3=number of terms in the expansion of this 'g'. #4=number of the molecular fit A1 function combination this 'g' belongs to.

`na1cof`

Length of the arrays numcom and cofcom, see next

`numcom`

Numcom (and cofcom) consists of a sequence of smaller sub arrays. Each sub array gives the expansion of a 'g' function in terms of the Cartesian functions in the pertaining fit function set. The elements of numcom specify the particular Cartesian sub functions that participate in the expansion. Its values are therefore limited to lie between 1 and (L+1)(L+2)/2, where L is the maximum *l*-value occurring in the fit function sets.

`cofcom`

Compare numcom: cofcom gives the actual expansion coefficients for the expression of a 'g' function in primitive Cartesian fit functions.

Section Num Int Params

Numerical integration parameters: the general precision parameter, but also more technical parameters used by the grid-generating modules.

`method`

Label of the method used to generate the grid. Usually: 'polyhedra'

`accint min`

Minimum integration precision parameter. It is the lower bound of the range in which the value of the actual numerical integration precision parameter may vary.

`accint max`

Maximum value of the precision general parameter

`accint`

Actual value of the precision parameter. This variable governs by default almost all other integration parameters.

`ldim`

In fact, this a geometric parameter: the number of dimensions in which the system is periodic. For molecules this is zero.

`PointChargeTypes`

The number of point charges types used in the calculation. Point charges belong to a different point charge type if, and only if, their strengths are not equal.

`accsph`

The precision parameter that determines the (radial) integration grid in the atomic spheres

`accpyr`

The precision parameter that determines the general precision level of the grid in the atomic polyhedra

`accout`

The precision parameter that determines the general precision level of the grid in the outer region

`accpyu`

The precision parameter that determines the 1D grid along the first direction in the quadrangles and triangles of the bases of the atomic pyramids

`accpyv`

The precision parameter that determines the 1D grid along the second direction in the quadrangles and triangles of the bases of the atomic pyramids

`accpyw`

The precision parameter that determines the 1D radial integration in the atomic pyramids, between the atomic sphere surfaces and the pyramid basis

`frange`

Estimated maximum range of functions, to determine how far the integration grid has to extend outwards, away from the molecule

`rspher`

An array with the radii of the atomic sphere (a value per atom type)

`rsph0`

The smallest sphere radius

`rsphx`

The largest sphere radius

`dishul`

The distance between the innermost boundary planes, which separate the atomic pyramids from the outer region, and the surfaces of the outermost atoms

`nouter`

The number of intervals in which the outward (radial) integration in the outer region is broken up

`outrad`

The precision parameter that determines the outward radial integration in the outer region

`outpar`

The precision parameter that determines the 2D integrals in the outer region parallel to the boundary planes

`linteg`

An array with maximum angular momentum quantum numbers (one value per atom type), to determine the angular integration grid in the atomic spheres

`lintgx`

Maximum of linteg()

`linrot`

Angular momentum quantum number to determine the rotational integration parameter around the molecular axis (in linear molecules only)

`ntyps`

The number of atom types as seen by the numerical integration grid generator. This means in practice: the number of non-dummy atom types plus the number of point charge types.

`nnucs`

The number of atoms as seen by the numerical integration grid generator. This means in practice: the number of non-dummy atoms plus the number of point charges.

`qatm`

Nuclear charges for all ntyps atom types

`nratst1`

The numerical integration grid generator automatically determines the symmetry of the nuclear (nnucs atoms!) frame and then puts the atoms in sets of symmetry equivalent ones. nratst1() is an array (0:ntyps) that contains the cumulative number of atoms in the symmetry sets. nratst1(k) is the total number of atoms in the sets up to and including set #k

`xyzatm`

Cartesian coordinates of the atoms.

`linteg all`

Similar to array linteg(), extended to include also the point charge types

`npowx`

Maximum power of the radial variable *r*, in the set of test functions that the grid generator uses to tune the grid

`alfas`

An array that stores the exponential decay factors of all test functions, ordered by atom type and by the power of the radial variable *r*.

Section Symmetry

Symmetry related data.

`nogr`

The number of symmetry operators in the point group used in the calculation. NB, for the special cases of infinite symmetries, only the operators corresponding to finite elements are counted. Therefore, ATOM has nogr=1 (only the unit operator); C(LIN) has nogr=1, D(LIN) has nogr=2.

`faith`

An array that stores all the (3,3) symmetry operator matrices in the real space representation

`nsetat`

The number of sets of symmetry equivalent atoms under the used symmetry

`napp`

An array that stores for each atom the number of the symmetry set it belongs to

`notyps`

An array that stores for each set of symmetry equivalent atoms, the atom type to which the set belongs

`noat`

Map between the normal list of atoms and the symmetry sets. When you loop over the symmetry sets and, inside, loop over the atoms in each set, you thereby run over the index of noat(). The value points to the position of that atom in the original (not set-ordered) list.

`ntr`

An array (nogr,nnuc) that stores for the each atom A and each symmetry operator R, the atom onto with A is mapped by R. The row index runs over all symmetry operators, the column index over the atoms.

`npeq`

The number of symmetry unique pairs of atoms

`jjsym`

An array that runs over the npeq sets of symmetry equivalent atom pairs. Its value gives for the indicated set the index of a (c.f. the first) atom pair in that set.

`jasym`

An array that runs over the npeq sets of equivalent atom pairs. Its value gives for the indicated the set the number of pairs in that set.

`ja1ok`

An array (1:npeq), with values 0 or 1. 1=the pair density can be fitted using A1 fit functions only. 0=all fit functions (on the involved atoms) are to be used. The value 1 may arise because of symmetry properties, or because the distance between the atoms is so large that the inaccuracy from using only A1 fit functions can be neglected.

`ntr_setat`

A condensed variety of array ntr: the columns are not the atoms, but the nsetat sets of symmetry equivalent atoms. The value is the index of the atom, onto which a representative (c.f. the first) atom of the indicated symmetry set is mapped by the given symmetry operator.

`igr`

A code that fixes, together with nogr and ngr, the point group symmetry. See the header of routine adf/maisya for a list

`ngr`

One of the code components that fix the symmetry group. See routine adf/maisya

`grouplabel`

Schönfliess symbol as used in ADF

`nsym`

The number of symmetry representation (including subspecies) used in the calculation.

`norb`

For each of the nsym representations the number of basis function combinations (SFOs) that belong to it.

`nfcn`

For each of the nsym representations the number of primitive atom centered basis functions that participate in the representation.

`ncbs`

For each of the nsym representations the number of core orthogonalization functions that participate in the representation.

`jsyml`

For each of the nsym representations: if it belongs to a one-dimensional irrep, the value is 1, otherwise: for the first subspecies in the irrep the value is the dimension of the irrep, for the other subspecies in the same irrep the value is 0

`symlab`

For each of the nsym representations the label (string) of the representation

`norboc`

An array (-2:2,nsym). The column runs over the symmetry representations. The positive row indices (1,2) specify for spin-A and spin-B (the latter only if the calculation is spin-unrestricted), the highest non-empty

orbital. The negative indices (-1,-2) specify for spin-A and spin-B (if the unrestricted fragment option is used) the total number of non-empty SFOs. The zero row index specifies the number of non-empty SFOs, before applying any fragment occupation changes.

Section Spin_orbit

(To be completed)

Section Energy

`XC energies`

16 elements of an array enxc(2,2,4): exchange-correlation energies of various charge densities:
first index: 1=exchange term, 2=correlation term
second index: 1=lda tern, 2=gga term
third index: 1=energy of fragments (summed over fragments), 2=energy of sum-of-fragments density, 3=energy of orthogonalized fragments, 4=SCF.

`Pauli TS Correction (LDA)`

Correction to the 'Transition State' method to compute terms in the bonding energy, in this case the Pauli exchange energy term. The Pauli TS Correction is not separately printed in the standard output file, but included in the Pauli interaction term.

`Pauli FitCorrection`

The first-order correction to the Pauli exchange interaction term, for the error in the Coulomb energy due to the fit incompleteness. This correction term is not printed in the output file but included in the Pauli interaction term

`Elstat Core terms`

An obsolete variable, not used in the energy computation

`Elstat Fitcorrection`

The first-order correction to the electrostatic interaction term (putting the fragments together, without any relaxation of Pauli orthogonalization), for the error in the Coulomb energy due to the fit incompleteness

`Orb.Int. FitCorrection`

The first-order correction to the electrostatic interaction term in the SCF relaxation energy (Orbital Interactions), for the error in the Coulomb energy due to the fit incompleteness. This term is not printed (anymore) separately, but incorporated in the symmetry-specific interaction terms.

`Orb.Int. TSCorrection (LDA)`

The difference between the representation-specific orbital interaction terms added, and a straightforward computation of the SCF relaxation energy is the result of the neglect of higher order terms in the Taylor expansion that underlies the 'Transition State' method. This difference, therefore, corrects exactly this neglect. It is not printed separately anymore in the output, but incorporated in (distributed over) the representation-specific orbital interaction terms.

`Ebond due to Efield`

Bond energy term due to any homogeneous electric field

`Corr. due to Orthogonalization`

For analysis purposes, the concept of 'orthogonalized fragments' has been introduced and the bonding energy is split in a part that describes the difference between the sum-of-fragments situation and the orthogonalized-fragments density at the one hand, and the SCF relaxation (from the orthogonalized fragments density) at the other. Both terms contain a first order fit correction term. The result of adding the two parts is not identical to computing the total bonding energy directly and applying the first order correction to that approach. The difference is given by this term, which therefore corrects for the additional second order fit errors caused by using the orthogonalized fragments split-up

`SumFragmentsSCF FitCorrection`

The 'true' first order fit correction for the complete bonding energy, resulting from a direct calculation that takes the sum-of-fragments as starting point and the SCF as final situation, without the intermediate step of orthogonalized fragments.

`Pauli Efield`

The contribution to the Pauli interaction energy due to any electric field

`Orb.Int. Efield`

The contribution to the SCF relaxation energy (orbital interactions) due to any electric field

`Electrostatic Interaction`

The electrostatic interaction energy including any first order fit correction (if computed from the fit density)

`Pauli Total`

The Pauli exchange (orbital orthogonalization) interaction energy

`Steric Electrostatic`

INCORRECT. Do not use. The electrostatic interaction energy including any first order fit correction (if computed from the fit density)

`Steric Total`

The total steric interaction energy, consisting of the electrostatic and the Pauli interactions

`Orb.Int.` *Irrep*

Irrep stands for one of the irreps of the point group symmetry. The value gives the orbital interaction (SCF relaxation) term for that symmetry representation

`Orb.Int. Total`

The total orbital interaction energy

`SCF Bond Energy`

Total bonding energy

`elstat`

INCORRECT. Do not use. Electrostatic interaction energy. Same as the 'Electrostatic Interaction' variable in this section

`Bond Energy`

Total bonding energy, same as the 'SCF Bond Energy' variable

`Pauli Kinetic`

Kinetic energy term in the Pauli exchange interaction energy

`Pauli Coulomb`

Coulomb energy term in the Pauli exchange interaction energy

`Pauli Kinetic+Coulomb`

Sum of the kinetic and Coulomb terms in the Pauli exchange interaction energy

Section Point_Charges

`NumberofPointCharges`

The total number of point charges used

`PointCharges`

The array with point charge values: (4,np), where np is the number of point charges and the 4 components are, respectively, the x y z components and the strength.

Section GeoOpt

Optimization data.

Where references are made to the list of atoms, the atoms are assumed to be in internal order. This may be different from the input-list of atoms.

`nfree`

number of independent optimization variables

`idfree`

indices (3,nr-of-atoms) for all atomic coordinates referring to the optimization variables (values 1..nfree) and/or LinearTransit parameters (values nfree+k, k being the k-th LT parameter). A zero value means that the coordinate is frozen.

`all freedoms`

A logical the flags whether or not all fundamental degrees of freedom in the system are allowed to vary. This is not the case when constraints are applied.

`Gradients`

The most recent values for the derivatives of the energy with respect to the atomic coordinates (cartesian or z-matrix, depending on the type of optimization variables).

`Displacements`

The most recent step executed for the atomic coordinates (optimization variables)

`kmatrix`

The connection matrix.

`Hessian_CART`

The Hessian matrix (second derivatives) as a n*n matrix, in the Cartesian coordinates representation. Note that the reduced storage mode (typically, Fortran upper-triangular) is not applied.

`Hessian_ZMAT`

Same, but now in the internal coordinates representation

`Hessian inverted_CART`

The *inverted* Hessian, in Cartesian coordinates

`Hessian inverted_ZMAT`

Likewise, in internal coordinates

Note: in most cases only one, or maybe two of the Hessian cases are present on TAPE13. They can be transformed into each other quite easily.

`xyz old`

cartesian coordinates at previous geometry cycle

`zmatrix old`

idem for internal coordinates

Section TS

Information about the Transition State search

`modtrc`

Defines the initial search direction. Positive value *n*: the n-th Hessian eigenvector (default:1). Negative value *n*: the Hessian eigenvector with the largest (absolute value) component in the n-th optimization variable

`itrace`

Index of the Hessian eigenvector that is being followed

`neghes`

The assumed number of negative eigenvalues of the Hessian at the Transition State. Should be 1: searches for higher-order transition states are not supported.

`mode to follow`

Direction vector in atomic coordinates (Cartesian or Z-matrix, depending on the variable geocrd) that corresponds to the current estimate of the unique Hessian eigenvector with negative eigenvalue

Section LT

Information about the Linear Transit calculation

`nr of points`

The total number of LT points to be computed.

`current point`

Index of point that is currently computed

`Energies`

Energy values in the LT points

`Dipole`

Dipole moments in the LT points

`Parameters`

LT parameters, initial and final values (along the path, the values are obtained by even-spaced linear interpolation)

`atmcrd`

ZMAT if a Z-matrix structure (connection matrix) is available. CART otherwise. Used for printing

`geocrd`

Type of coordinates to optimize and scan along the path (CART or ZMAT)

`xyz`

Cartesian coordinates in the LT points (3,natoms,nlt)

`zmatrix`

Internal coordinates in the LT points

`AtomCharge Mulliken`

Mulliken atomic charges in the LT points

`FragmentCharge Hirshfeld`

Hirshfeld fragment charges in the LT points

`AtomCharge_initial Voronoi`

Voronoi atomic charges corresponding to the sum-of-fragment densities in the LT points

`AtomCharge_SCF Voronoi`

Voronoi atomic charges corresponding to the SCF densities in the LT points

Section IRC

This section contains general information about the IRC (Intrinsic Reaction Coordinate) calculation. Details of the computed reaction path are in sections IRC_Forward and IRC_Backward.

`atmcrd`

ZMAT is a Z-matrix structure (connection matrix) is available. CART otherwise

`geocrd`

CART or ZMAT: the type of coordinates to change, optimize and trace

`PointStatus`

A string status variable of the current IRC point. Value can be 'DONE' (if the point has been computed), 'EXEC' if its computation has not yet finished.

`nfree`

Number of optimization coordinates that can be varied. See section GeoOpt

`idfree`

(3,natoms) pointers to the optimization variables for each of the atomic coordinates. A zero means: frozen by constraint

`xyz`

Cartesian coordinates

`kmatrix`

Connection matrix, if a Z-matrix structure is available

`zmatrix`

Internal coordinates

`Energies`

Energy at the Transition State

`Dipole`

Dipole moment at the Transition State

`Gradients`

Computed energy gradients at the (assumed) Transition State (should be very small)

`AtomCharge Mulliken`

Mulliken atomic charges, for the TS geometry

`AtomSpinDen Mulliken`

Atomic spin densities (Mulliken) at the TS

`AtomCharge_initial Voronoi`

Voronoi atomic charges at the TS, from the sum-of-fragments density

`AtomCharge_SCF Voronoi`

Similar, for the SCF density

`modtrc`

Defines the start direction for the IRC path. A positive value *n* selects the n-th eigenvector of the Hessian. A value -1 selects the gradient vector (which must then, of course, not be exactly zero). A value -2 specifies that the start direction is specified in the input file.

`step`

Step length (in mass-weighted metric) between successive points of the IRC path.

`stepMin`

The minimum value for the step

`stepMax`

The maximum value for the step

`Hessian inverted_ZMAT`

Inverse Hessian in internal coordinates.

`lfree`

The number of independent optimization step directions (for the restricted optimization orthogonal to the IRC path.

`vfree`

Direction vectors (3,natoms,lfree) for the independent optimization directions

`GradientVector`

The current gradient vector (during optimization)

Section IRC_Forward

Information about the 'forward' IRC path. The choice, which direction down from the Transition State is forward or backward is arbitrary. By definition, in ADF the forward direction is in the positive direction along the first Hessian eigenvector, for which the sign convention is that the largest coefficient is positive.

`PathStatus`

Status (string) variable for the 'forward' half of the IRC path. May be 'EXEC', or 'DONE', 'UNKNOWN', 'WAIT', 'OFF'

`PointStatus`

Status variable for the current point at the 'forward' path. May be 'DONE', 'EXEC'

`nset`

Size of arrays to store data in the IRC points along the path. Will be increased when too small

`pivot`

Coordinates of the current pivot point

`xyz`

Cartesian coordinates of the IRC points (3,natoms,nset)

`zmatrix`

Internal coordinates of the IRC points (3,natoms,nset)

`Path`

Lengths measures in mass-weighted metric along the path to the IRC points

`Curvature`

Local curvature values of the path at the IRC points

`Energies`

Energy values at the IRC points

`Gradients`

Energy gradients at the IRC points (one value: the gradient along the path. The orthogonal components are presumably zero)

`Dipole`

Dipole moments at the IRC points

`AtomCharge Mulliken`

Mulliken atom charges at the IRC points

`FragmentCharge Hishfeld`

Hirshfeld fragment charges at the IRC points

`AtomCharge_initial Voronoi`

Voronoi atomic charges at the IRC points, corresponding to sum-of-fragments densities

`AtomCharge_SCF Voronoi`

Voronoi atomic charges at the IRC points, corresponding to the SCF densities

`CurrentPoint`

Integer index of the current IRC point (in the set of nset points)

`step`

Current step length

Section IRC_Backward

All entries in this section match those in the section IRC_Forward. Of course, here they refer to the other half of the IRC path.

Section Freq

This section contains information about (progress) of the Frequencies calculation and results.

`kountf`

An integer counter that keeps track of how many geometric displacements have been carried out to scan the potential energy surface around the equilibrium

`nraman`

Integer to flag whether or not Raman intensities are computed

`numdif`

Integer to determine the type of numerical differentiation (of gradients, to get the second derivative): 1=one-sided, 2=two-sided displacements.

`disrad`

Size of displacements of Cartesian coordinates or bond lengths (in case of displacements in internal coordinates)

`disang`

Size of displacements of angular coordinates

`geocrd`

Type (string) of coordinates to displace: CART or ZMAT

`atmcrd`

ZMAT if a z-matrix structure is available. This determines printed output but does not affect the computation. Compare the variable geocrd

`nfree`

The number of degrees of freedom

`idfree`

An array (3,natoms) that stores for each atomic coordinates (Cartesian or internal, depending on geocrd) the number of the (1..nfree) variational freedom it corresponds to. If zero, the coordinate is frozen by constraint.

`xyz`

Cartesian coordinates of the equilibrium geometry

`kmatrix`

Connection matrix that defines a Z-matrix

`zmatrix`

The Z-matrix variable values of the equilibrium geometry

`all freedoms`

Logical: true if all atomic coordinates are allowed to be displaced, not restricted by constraints.

`nr of atoms`

The total number of atoms, including dummy's

`rigids`

Vectors of rigid motion directions, expressed in the atomic coordinates (3,natoms,6)

`Dipole previous`

The dipole moment of the previous geometry. This is used to compute dipole derivatives by numerical differentiation. The 'previous' geometry is the equilibrium geometry in case of one-sided displacements.

`Dipole`

The dipole moment corresponding to the current geometry

`Dipole derivatives`

The matrix of dipole derivatives with respect to atomic displacements

`Polbty previous`

The polarizability tensor (6 elements, triangular representation) of the 'previous' geometry. See the remarks about the dipole moment

`Polbty`

The polarizability tensor corresponding to the current geometry

`Polbty derivatives`

The matrix of derivatives (w.r.t. the atomic coordinates) of the polarizability tensor

`Gradients`

The energy gradients corresponding to the current geometry

`Gradients previous`

The energy gradients of the 'previous' geometry. See the remarks about 'previous' dipole moment

`Force constants`

The matrix of force constants (second derivatives), built up during the frequencies calculation.

`xyz displaced`

The Cartesian coordinates of the current (displaced) geometry

`zmatrix displaced`

Internal coordinates of the current (displaced) geometry

`Dipole derivatives_CART`

Dipole derivatives with respect to Cartesian coordinate changes

`Hessian_CART`

The Hessian matrix in Cartesian coordinates, computed at the end, when the construction of the 'Force constants' has been completed.

`Frequencies`

An array with harmonic frequencies.

Sections Ftyp n

n is an integer. All such sections give general information about fragment type #n, and more specifically about the adf calculation that produced the corresponding fragment file.

`jobid`

Job identification of the fragment run

`title`

Title of that calculation

`nsym`

Number of symmetry representations (subspecies) used

`norb`

For each representation the size of the Fock matrix (variational degrees of freedom)

`bb`

Labels of the subspecies

`igr`

(Partial) code for the point group symmetry

`ngr`

(Partial) code for the point group symmetry

`grouplabel`

Schönfliess symbol of the point group symmetry (of the fragment calculation)

`nfcn`

An array over the representation: for each subspecies the number of primitive STO basis functions that participate in that subspecies

`jsyml`

An array (1:nsym). Value 1 means that the corresponding subspecies belongs to a 1D irrep. A value larger than 1 means a correspondingly higher dimensionality of the irrep *and* indicates that that subspecies is the first in that irrep. A value 0, finally, means that it is not the first subspecies in its irrep.

`nfrag`

Number of fragments used in that fragment calculation

`natom`

Number of atoms in the fragment

`naos`

Number of primitive atomic basis functions

`nrat 1`

Maps the atoms of this fragment (the '1' signals the first fragment of this type) onto the list of all atoms

`rotfrg`

Rotation matrix to map the fragment coordinates as they are on the fragment file onto their actual orientation in the molecule

`nsot`

Total number of MO degrees of freedom, summation over all subspecies

`nmis`

The number of symmetry representations that could not be spanned by the basis set

`mis`

Indices of the missing symmetry representations

Sections Ftyp n?

n stands for the n-th fragment type. The ? stands for one of the symmetry representations (of the point group symmetry used in the fragment calculation)

`froc`

MO occupation numbers for the MOs in this subspecies

`eps`

Orbital energies
When they result from a ZORA calculation, the non-scaled values are stored on file (the scaled values are printed in the standard output file).

`eigvf`

Fragment MO eigenvectors, expressed in all the primitive atomic orbitals of the fragment.

`nsos 1`

Total number of MOs in this subspecies: size of variational problem

`nbas 1`

Number of primitive atomic basis functions that participate in this subspecies

`npart 1`

Indices that give for each of the nbas functions, the number of the basis function in the list of all basis functions

`FO 1`

The fragment MOs (nbas*nsos coefficients)

`nocc 1`

Number of non-empty orbitals

Section Freq Symmetry

Information about the true (possibly input-specified) symmetry of the equilibrium geometry (in a frequencies calculation). The displaced geometries may loose symmetry. Therefore, the program uses NOSYM symmetry, internally, for a frequencies calculation. The 'true' symmetry of the system is used for analysis purposes.

`nr of operators`

Number of symmetry operators used

`operators`

(3,3) matrices of the operators

`nr of symmetries`

Number of subspecies

`symmetry labels`

Names of the subspecies

`atom indices`

List of indices to map the symmetry-ordered atoms (loop over symmetry sets, loop over atoms in each set) to the 'normal' list of all atoms

`nr of atomsets`

Number of sets of symmetry equivalent atoms

`atom mappings`

Integer array that provides mapping (back and forth) between the atom list in the input file and the internally used list, which is atom type driven

`atomset indices`

The number of atoms in each of the sets of symmetry equivalent atoms

`nr of displacements_X`

(X must be one of the symmetry representations.) The number of symmetry-combined atomic displacements that transform as X

`degeneracy_X`

Degeneracy of X

`displace_X`

The actual displacement direction vectors (3,natoms,N). N is the number of symmetry displacements for X.

`nr of rigids_X`

The number of rigid motion direction vectors that transform as symmetry representation X

`displ_InputOrder_X`

The displacement vectors, but now expressed in the atomic coordinates using the ordering of atoms in the

input file

`NormalModes_X`

Harmonic frequency normal modes in representation X

`Frequencies_X`

The harmonic frequency values

`IR intensities_X`

The infrared intensities

Sections X

X stands here for the label of a subspecies of the point group symmetry, for instance A1. Depending on the point group symmetry, there may be many such sections, each corresponding to one of the subspecies. All such sections have an identical structure.

`nmo_A`

The number of MOs with spin-A, for which the coefficient vectors are calculated. During the SCF this may be severely reduced, at the end it is usually the complete basis in the pertaining symmetry representation.

`nmo_B`

Similar for spin b. This variable is not present in a restricted calculation.

`SFO`

The definition of the SFOs in the representation, consisting of expansion coefficients in terms of the primitive atomic STO basis functions

`frocf`

The occupation numbers of the SFOs in this representation

`npart`

A list of indices of the bas functions that are used in this symmetry representation

`froc_A`

The occupation numbers of the MOs in the representation, for spin-A

`froc_B`

Similar for spin-B, if a spin-unrestricted calculation is performed

`smx`

Overlap matrix between core functions and SFOs

`frocor`

SFO occupation numbers

`Orth-Bas`

The orthogonalized fragment orbitals in the BAS representation.

`Low-Bas`

The Lowdin orbitals in the BAS representation: the matrix to transform the MOs from Lowdin representation (orthonormalized SFOs) to the BAS representation

`Eigen_Bas_A`

mo expansion coefficients in the bas representation for all nmo_A orbitals. The coefficients run over all bas functions indicated by npart

`Eigen_Bas_B`

Similar for spin-B, if present

`eps_A`

The orbital energies for the nmo_A orbitals of spin-A
When they result from a ZORA relativistic calculations, the non-scaled values are stored on file. (The scaled energies are printed in standard output.

`eps_B`

Similar for spin-B, if present

`Eig-CoreSFO_A`

MOs expressed in SFOs, for spin-A MOs

`Eig-CoreSFO_B`

Similar for spin-B

Sections Atyp n X

Each such section contains the (core- and possibly also valence-) radial density and potential of one particular atom type. *X* is the atom type label and *n* is an index running over all atom types in the calculation. The list of all atom types is printed on standard output in the early geometry section.

The radial densities and potentials may be represented as simple tables - a sequence of values for *r*, the distance to the nucleus, and the corresponding density or potential - or as a piecewise expansion in Chebyshev polynomials over a sequence of intervals (r1,r2).

The core density and potential have been constructed from the Frozen Core orbitals, which are defined in the section Core. If a TAPE12 (corepotentials) file has been attached to the calculation the core data is read off from that TAPE12 and stored also.

`rx val`

Maximum r-value for which the valence density is non-negligible

`nrint val`

Number of intervals for piecewise expansion of the valence density in Chebyshev polynomials

`rup val`

Arrays (1..nrint) of upper bounds of the intervals. The lower bound of the first interval is zero

`ncheb val`

Array (1..nrint) with the number of expansion coefficients for each interval

`ccheb val`

Coefficients of the expansion. All coefficients, for all intervals, are stored contiguously in one linear array. The parts pertaining to a particular interval are determined by using the arrays ncheb()

`nrad`

Number of points used in the direct tabular representation of the atomic densities and potentials

`rmin`

The first r-value of the table: the radial grid is defined by a first value (rmin), a constant multiplication factor defining rk+1 w.r.t. rk (rfac, see next), and the total nr of points (nrad).

`rfac`

The multiplication factor of the radial grid

`valence den`

The valence density, in a table of nrad values.

`valence pot`

Similar for the Coulomb potential of the density, including a nuclear term Q/r, such that the long-range monopole term in the potential is zero

`qval`

The number of electrons contained in the valence density

`rx core`

Maximum r-value for which the core density is non-negligible

`nrint core`

Number of intervals for piecewise expansion of the core density in Chebyshev polynomials

`rup core`

Arrays (1..nrint) of upper bounds of the intervals. The lower bound of the first interval is zero

`ncheb core`

Array (1..nrint) with the number of expansion coefficients for each interval

`ccheb core`

Coefficients of the expansion. All coefficients, for all intervals, are stored contiguously in one linear array. The parts pertaining to a particular interval are determined by using the arrays ncheb()

`qcore`

The number of electrons contained in the core density

`core den`

The core density, in a table of nrad values.

`core pot`

Similar for the Coulomb potential of the density, including a nuclear term Q/r, such that the long-range monopole term in the potential is zero

Section LqbasxLqfitx_xyznuc

This section will be removed again in the future. Temporarily it serves to transfer data from the calling program to the grid generator.

`lqbasx`

An array with for each atom type the maximum angular moment quantum number in the basis functions for that type

`lqfitx`

An array with for each atom type the maximum angular moment quantum number in the fit functions for that type

`xyznuc`

Cartesian coordinates of the non-dummy atoms

Section GenptData

This section will be removed in the future. It serves, temporarily, to transfer data from the calling program to the numerical integration grid generator. Most of the entries here occur also in other sections but are packed together as replacement for previous common block structure.

`numint`

Integer code for the type of integration grid. Usual value: 2 (polyhedra method)

`iexcit`

Integer flag for excitations (response) calculation

`lpolar`

Integer flag for polarizability (response) calculation

`ldim`

Number of dimensions of periodicity

`mdim`

Dimensionality of the molecule, for instance a linear molecule has mdim=1

`r0mult`

A technical parameter that sets the radius outside which the multipole part of the fit coulomb potential functions is separated (from the exponentially decaying part), for separate treatment in the evaluation of the molecular coulomb potential.

`avec`

(3,3) matrix with lattice vectors. Only the (ldim,ldim) sub matrix is significant.

`bvec`

Inverse of avec (apart from a factor of 2 pi): lattice vectors in reciprocal space.

`ngimax`

Maximum number of geometry optimization iterations

`llbloc`

Block length determination parameter (maximum)

`ipnbl`

Number of integration blocks processed by the current process

`nbleqv`

The number of symmetry equivalent blocks to each symmetry unique block of points. This value is 1 if any equivalent blocks are not constructed and used.

`ngmax`

The number of integration points, accumulated over all parallel processes

`nblock`

The number of integration blocks

`lblock`

The block length

`lblx`

An upper bound of the block length applied during the computation of the block length

`nmax`

The number of integration points generated by this process

`twopi`

Value of the constant 2**ΤΤ**

`fourpi`

Value of the constant 4**ΤΤ**

Section Multipole matrix elements

Information in a response calculation

`dipole elements`

The matrix elements of the 3 dipole operator components between occupied and virtual orbitals: outer loop

over the operators (in order: y, z, x), loop over virtual MOs, inner loop over occupied MOs

`quadrupole elements`

Similar as for dipole. Order of operators:
√(3)*xy
√(3)*yz
z2-(x2+y2)/2
√(3)*xz
√(3)*(x2-y2)/2

`octupole elements`

Similar as for dipole and quadrupole. Order of operators:
√(10)*y*(3*x2-y2)/4
√(15)*xyz
√(6)*y*(4*z2-x2-y2)/4
z*(z2-3(x2-y2)/2)
√(6)*x*(4*z2-x2-y2)/4
√(15)*z*(x2-y2)/2
√(10)*x*(x2-3y2)/4

`hexadecapole elements`

Similar as for dipole and quadrupole. Order of operators:
√(35)*xy*(x2-y2)/2
√(70)*z*(3x2y-y3)/4
√(5)*xy*(6z2-x2-y2)/2
√(10)*(4yz3-3yz*(x2+y2))/4
(8z4-24*z2*(x2+y2)+3(x4+2x2y2+y4))/8
√(10)*(4xz3-3xz*(x2+y2))/4
√(5)*(x2-y2)*(6z2-x2-y2)/4
√(70)*z*(x3-3xy2)/4
√(35)(x4-6x2y2+y4)/8

Section Irreducible matrix elements

Information in a response calculation

`irreducible dipole elements`

The dipole matrix elements between occupied and virtual MOs, as in the section Multipole matrix elements. Here, however, the matrix elements are ordered by symmetry representations and 'symmetry zeros' are omitted. The stored arrays, however, have the same size as in the previous section. See the implementation for details about the storage of this data. (Directory $ADFHOME/adf/response/)

`irreducible quadrupole elements`

Similar as for the dipole elements

`irreducible octupole elements`

Similar as for the dipole elements

`irreducible hexadecapole elements`

Similar as for the dipole elements

Section ETS

Technical data used in the ets procedure.

`nff`

Size of array ncspt (next)

`ncspt`

Pointer array to find, for each atom type, the first element corresponding to that atom type's section in the arrays ncsett, alfcst, and cfcset, see below

`ncs`

Size of the matrices ncsett, alfcst, and cfcset, see below

`ncsett`

Build a list of products of core orbital expansion functions, taking only the one-center products and looping over the atom types (not the atoms). ncsett stores the powers of the radial variable *r* for the products (from the main quantum numbers, one subtracted). A product of a 1s and a 2p yields ncsett()=1

`alfcst`

Similar as ncsett: the sum of the exponential decay factors of the factor functions

`cfcset`

The density matrix corresponding build from the frozen core orbitals (all atom types, but no copies for the distinct atoms of a type), in the representation of the core orbital expansion functions. Stored are, per atom and per *l*-value (0..3) the upper-triangles of the corresponding density matrices, one after the other, all in cfcset

`nnuc`

The number of (non-dummy) nuclei

`qcore`

For each atom the number of electrons summed over its core orbitals, resulting from analytical integration of the core orbital expansions in STO core expansion functions.

Using Data from TAPE21

An ASCII dump of TAPE21 (complete or partial) can be obtained with the kf utility dmpkf, see the utilities document. Alternatively you may build your own small program to extract any required information, using the KF library routines in the adf package. Consult the KFS documentation for a description of this software.

Representation of functions and frozen cores

adf uses the cartesian representation for the spherical harmonics part in functions:

f(x,y,z)=xaybzcrde-ar

The angular momentum quantum number *l* is then given by *l*=a+b+c, and the main quantum number *n*=*l*+d+1.

There are (*l*+1)(*l*+2)/2 different combinations of (a, b, c) for a given *l*-value, rather than (2*l*+1). The excess is caused by the presence of spurious non-*l* Functions in the set; a Cartesian d-set for instance consists of six functions, five of which are true d-functions while one linear combination is in fact an s-type function (x2+y2+z2). Only the five true d-combinations are actually used as degrees of freedom in the basis set, but lists of primitive basis functions (bas) for instance run over all Cartesian functions including the improper ones.

A function *set* in adf is characterized by the quantum numbers *l* and *n*, and by the exponential decay factor a. A set thus represents (*l*+1)(*l*+2)/2 Cartesian functions and (2*l*+1) degrees of freedom.

The atomic frozen core orbitals are described as expansions in Slater-type functions; these are not the functions of the normal basis set but another set of functions, defined on the data files you use in Create mode.

Orthogonality of the valence space to the frozen core states is enforced as follows: for each frozen core shell (characterized by the quantum numbers *l* and *n*: all orbitals with *m*=-*l*...+*l* are identical apart from rotation in space) the set of valence basis functions is augmented with a so-called core orthogonalization function set. You may conceptually interpret the core orthogonalization functions as single zeta expansions of the true frozen core states. Each of the normal valence basis functions is now transformed into a linear combination of that valence function with all core orthogonalization functions, where the coefficients are uniquely defined by the requirement that the resulting function is orthogonal to all true core functions.

So the list of all Cartesian basis functions is much larger than the degree of freedom of the basis: it contains the spurious *non-l* combinations and it contains also the core orthogonalization functions.

Evaluation of the charge density and molecular orbitals

TAPE21 contains all the information you need to evaluate the charge density or a Molecular Orbital (MO) in any point in space. Most of the information is located in section Basis:

A list of function characteristics (kx,ky,kz,kr,alf), including the core orthogonalization functions. This list does *not* run over *all* bas functions used in the molecule: if a particular function is used on the atoms of a given atomtype, the function occurs only once in the list, but in the molecule it occurs as many times as there are atoms of that type.

With array nbptr you locate the subsections in the function list that correspond to the different types of atoms: for atom type i the functions nbptr(i)...nbptr(i+1)-1.
The distinct atom types are listed in an early section of the standard output file.

Array nqptr gives the number of atoms for type i: nqptr(i+1)-nqptr(i). With this information you construct the complete list of all functions. Repeat the subsection of type i as many times as there are atoms of that type: the complete list can be considered to be constructed as a double loop, the outer being over the atom types, the inner over the atoms that belong to that type.
The total 'overall' list of functions you obtain in this way contains naos functions.
Note that in this way we have implicitly also defined a list of all atoms, where all atoms that belong to a particular atom type are contiguous. This list is the so-called 'internal' atom ordering, which may not be identical to the order in which atoms were specified in input, under atoms.

For a given symmetry representation (Sections S) the array npart gives the indices of the basis functions in the overall list that are used to describe orbitals in this representation.
In case of an unrestricted run the array npart applies for either spin: the same basis functions are used; the expansion coefficients for the molecular orbitals are different of course.

In the symmetry-representation sections Eigen_bas gives the expansion coefficients that describe the MOs. The expansion refer to the functions indicated by npart, and the function characteristics are given by the arrays kx,ky,kz,kr,alf, *and* bnorm, i.e. the expansion is in *normalized* functions.

The value of an MO is now obtained as a summation of values of primitive basis functions. For the evaluation of any such basis function you have to take into account that its characteristics are defined in the local coordinate system of its atom.

To obtain the charge density you sum all MOs, squared and multiplied by the respective occupation numbers (array froc in the appropriate irrep section).

Note that the auxiliary program densf, which is provided with the adf package, generates orbital and density values on a user-specified grid. See the utilities document.

4.5 TAPE13

TAPE13 is the checkpoint file for restarts after a crash. It is a concise version of TAPE12, containing only the data the program uses for restarting the calculation. See the restart keyword.

Like TAPE21, TAPE13 is a binary, keyword driven KF file. You can manipulate it with the KF utilities, to get a print-out of its 'table of contents', or a complete ASCII dump of its full contents.

The calculation that produces TAPE13 determines which section are written on it. The following sections may occur (and if they occur, the listed variables are stored in them). The actual values of the variables should be identical to the corresponding variables on TAPE21. Also they should have the same names and be located in the same sections. In some cases, TAPE13 contains the complete corresponding section of TAPE21.

Contents of TAPE13

Section Fit

`coef_SCF`

SCF fit coefficients. Total number of them is nprimf, the number of primitive fit functions (counting all *Cartesian* spherical polynomials: 3 for a *p*-set, 6 for a *d*-set, and so on). If the calculation is spin-unrestricted, each spin has its own set of fit coefficients: first all coefficients of spin-A, then those of spin-B

`coef_FreqCenter`

Only in a Frequencies calculation: the fit coefficients that correspond to the equilibrium geometry. The variable coef_SCF corresponds always to the current geometry, or the previous one if the geometry has just been changed and the new SCF has yet to start.

Section Freq

This section is identical to the same section on TAPE21.

Section Geometry

This section is identical to the same section on TAPE21

Section GeoOpt

This section is identical to the same section on TAPE21

Section IRC

This section is identical to the same section on TAPE21

Section IRC_Forward

This section is identical to the same section on TAPE21

Section IRC_Backward

This section is identical to the same section on TAPE21

Section LT

This section is identical to the same section on TAPE21

Section TS

This section is identical to the same section on TAPE21

5 RESULTS

5.1 Properties

Electronic Configuration, Orbital Energies

The direct results from the SCF are the orbital energies and occupation numbers. This defines the electronic configuration: the occupation numbers and HOMO and LUMO energies for instance show whether or not the aufbau principle is satisfied in the final situation.

The energies of the Core Orbitals can be used to interpret for instance XPS (X-ray Photoelectron Spectroscopy) data: from Koopman's theorem these core orbital energies are an approximation to the core ionization energies. This neglects the effect of relaxation upon the ionization so that absolute energy values may not be very good; relative values, however, should be fair and can therefore be used to study (relative) chemical shifts.

Populations and Atomic Charges

Mulliken populations

Mulliken-type populations are computed and printed at various levels of refinement (ranging from *per-basis function* to *per-fragment type*, data for the whole molecule as well as for individual MOs), and in two different representations, one based on the elementary basis functions (bas), the other on SFOs (Symmetrized Fragment Orbitals). This is potentially a very large amount of data. Precisely what is printed by default, and how this can be modified so as to suppress output or, alternatively, to get more information, is regulated by the print keys (print, eprint).

Atom charges, fragment charges, and bond orders

Mulliken populations can be summarized to yield atomic charges. Alternative methods exist to deduce atom charges from the self-consistent results of a molecular calculation. Three of those alternatives are provided by adf: Hirshfeld analysis, Voronoi analysis, and multipole derived charges.

The Hirshfeld analysis produces a charge value per fragment, computed as the integral of the SCF charge density over space, in each point weighted by the relative fraction of the (initial) density of that fragment in the total initial (sum-of-fragments) density:

$$Q^{\text{frag(i)}} = \int \rho^{\text{SCF}} \, \rho^{\text{initial frag(i)}} / (\sum_j \rho^{\text{initial frag(j)}}) \quad (5.1.1)$$

The Voronoi charge analysis consists of assigning the charge density in a point in space to the nearest atom. The Voronoi cell of an atom is the region in space closer to that atom than to any other.

This partitioning of space, using mid-way separation planes, is inappropriate to produce useful absolute numbers when neighboring atoms have very different sizes, for instance, Hydrogen and a heavy metal. However, *changes* in the density analyzed in this way do give a reasonable general insight in the effect of bonding on the location of charge densities, in particular because the Voronoi data per atom are split up in contributions within the atomic sphere and the rest of its Voronoi cell.

Hirshfeld and Voronoi charge analyses are printed at the end of the SCF (of the last geometry, in case of an Optimization).

The Hirshfeld analysis in adf produces charges *per fragment*, so that *atomic* charges are obtained only if single-atom fragments are used. This limitation does not apply to Voronoi charges (data per atom). Mulliken charges are given both per atom *and* per fragment.

In the printout of charges per fragment (as for the Hirshfeld analysis), you have to be aware of the *ordering* of fragments. A complete list of fragments is printed in the early GEOMETRY section of standard output, where you also find which atom(s) correspond(s) to which fragment. Note that even when you use single-atom fragments only, the order of fragments is usually quite different from the order of atoms in your input file. Typically (but not necessarily exactly in each case), when you use single-atom fragments: consider the first non-dummy atom in your ATOMS block. This defines the first atom *type*. Then browse the ATOMS list until you find an atom of a different type. This defines the second atom type, and so on. The single-atom fragment list will often be such that you first get *all* atoms of the first atom type, then all atoms of the second type, and so on. Check the printed list-of-fragments always, to avoid mistakes in assigning Hirshfeld charges to atoms (fragments).

The multipole derived charges (MDC) analysis [170] uses the atomic multipoles (obtained from the fitted density) up to some level X, and reconstructs these multipoles exactly (up to level X) by distributing charges over all atoms. This is achieved by using Lagrange multipliers and a weight function to keep the multipoles local. Since the atomic multipoles are reconstructed up to level X, the molecular multipoles are represented also up to level X. The recommended level is to reconstruct up to quadrupole: MDC-q charges.

Mayer bond orders are printed at the end of the SCF, see the key EXTENDEDPOPAN.

Bond order analysis

The bond order analysis produces the output in which the bond order values are printed for each pair of atoms for which the Nalewajski-Mrozek bond order value is larger than the threshold that can be specified with the keyword BONDORDER. For convenience the printed bond orders are accompanied by the corresponding inter-atomic distance. In the Nalewajski-Mrozek approach [148-153] the bond order indices b_{AB} are calculated based on the one- and two-center valence indices

$$b_{AB} = V_{AB} + w^{AB}_{A} V_A + w^{AB}_{B} V_B$$

with the weighting factors for one-center indices given by

$$w^{XY}_{X} = V^{cov}_{XY} / \sum_Z V^{cov}_{XZ}$$

Unlike other definitions of covalent bond orders, the Nalewajski-Mrozek valence indices comprise both, covalent and ionic contributions. There exist three alternative sets of the Nalewajski-Mrozek valence indices, [148-153, 140]. The bond order indices calculated from each set of the valence indices differ slightly due to arbitrariness in the way of splitting the one-center terms between bonds. More detailed description of alternative valence indices and their physical meaning is summarized in [148]; see also original papers [149-153]

By default the bond order indices based on the valence indices obtained from partitioning of $Tr(P\Delta P)$ are printed in the ADF output. Note that in this version the covalent two-center part (also printed in the output) is equal to the Gopinathan-Jug [153] bond order. The default values are:

$$V_A = V^{ion}_{A} + V^{cov}_{A}$$

$$V^{ion}_{A} = \sum_{a \in A} \{P^{\alpha}_{aa} \Delta P^{\alpha}_{aa} + P^{\beta}_{aa} \Delta P^{\beta}_{aa}\}$$

$$V^{cov}_{A} = 2 \sum_{a \in A} \sum_{a' \in A, a<a'} \{P^{\alpha}_{aa'} \Delta P^{\alpha}_{a'a} + P^{\beta}_{aa'} \Delta P^{\beta}_{a'a}\}$$

$$V^{cov}_{AB} = 2 \sum_{a \in A} \sum_{b \in B} \{P^{\alpha}_{ab} \Delta P^{\alpha}_{ba} + P^{\beta}_{ab} \Delta P^{\beta}_{ba}\}$$

To produce the values from all alternative versions of Nalewajski-Mrozek valence indices, accompanied by the Gopinathan-Jug [153] and Mayer [140] bond orders, see the keyword BONDORDER.

The Mayer [140] bond orders can also be calculated using the keyword EXTENDEDPOPAN. The two implementations of calculating the Mayer bond orders differ slightly if one uses frozen cores. They should agree exactly in all electron calculations.

Energy

The program prints the bonding energy (not in a Create or Frequencies run) and its decomposition in terms that are useful for chemical interpretation. The *total* energy is not computed. The bonding energy is defined relative to the fragments. When *basic atoms* are employed as fragments one should realize that these do not represent the atomic ground state since they are computed as spin-restricted and spherically symmetric objects, with possibly fractional occupation numbers. The correct multiplet state is not computed. To obtain the bonding energy with respect to isolated atoms you should therefore add atomic correction terms to account for spin polarization and the multiplet state. See also the SLATERDETERMINANTS key and the discussion in the Theory document on multiplet states.

The spin polarization energy can be computed by running the single atom unrestricted, using as fragment the corresponding (restricted) basic atom. The true multiplet state is not necessarily obtained in this way.

For the comparison of computed bonding energies with experimental data one should furthermore be aware of any aspects that are not represented in the computational formalism, such as zero-point motions and environment (solvent) effects.

In a Geometry Optimization or Transition State search, the program may print a bonding energy evaluation at each geometry (depending on print switches). A test-energy value is written in the log file. This is *not* the bonding energy, although the difference is usually small. The test-energy printed in the log file is the energy expression from which the energy gradients are computed. The true bonding energy contains in addition a few (small) correction terms that are mostly related to the fit incompleteness. These correction terms are usually very small.

If Electric Fields are used in the computation (homogeneous and/or point charges), the printed Bonding Energy is the energy of the molecule in the field minus the energy of the fragments in the same field. The energy terms due to the field are also printed separately so that one can subtract them from the total bonding energy to obtain the energy-change without field-terms.

Thermodynamics

At the end of a completed Frequencies calculation, a survey is given of thermodynamic properties: Heat Capacity, Internal Energy, Entropy. The computed results assume an ideal gas, and electronic contributions are ignored. The latter is a serious omission if the electronic configuration is (almost) degenerate, but the effect is small whenever the energy difference with the next state is large compared to the vibrational frequencies.

```
THERMO {P=pressure} {T=temp1 {temp2}} {nT=nT}
```

`pressure`

The Pressure in atmospheres. Default value: 1.0. A zero or negative pressure is adjusted by the program to a (very) small number 1e-4

`temp1,temp2`

The endpoints of the Temperature range (in K), for which the results are computed. By default only room temperature is considered (298.15 K).
If the option T= is used and only one value supplied (temp1), then temp2 is assumed to be equal to temp1.
A zero or negative temparture is adjusted by the program to a (very) small number 1e-4

`nT`

The number of steps by which the temperature interval is scanned. By default it is computed by the program from the temperature range (temp1, temp2), such that the step size is as close as possible to 10 K. Note that the number of temperatures for which the calculations are done is one more than the number of temperature *steps*.

The thermal analysis is based on the temperature dependent partition function. The energy of a (non-linear) molecule is

$$E/NkT = 3/2 + 3/2 + \sum_j^{3N-6}[\ h\nu_j/(2kT) + h\nu_j/kT(e^{h\nu_j/(kT}-1))\ ] - D/kT \quad (5.1.2)$$

The summation is over all harmonic ν_j, h is Planck's constant and D is the dissociation energy

$$D = D_0 + \sum_j h\nu_j/2 \quad (5.1.3)$$

5.2 Plots: Density, Potential, Orbitals

To compute the electrostatic potential, charge density or molecular orbital values in a regular 2-D or 3-D grid, a separate program densf can be used. It requires the TAPE21 result file from the calculation and produces a TAPE41 files with the required data.

Other programs may process the TAPE41 data.

Cntrs processes computes contours.

Adfplt displays orbitals, densities, potentials on your screen (2D, 3D) and can be used to print the pictures [137].

densf, cntrs and adfplt are auxiliary programs in the adf package. See the Analysis document.

6 APPENDICES

6.1 Database

The database contains standard basis sets (and fit sets, frozen core orbitals) for all chemical elements of the periodic table at different levels of accuracy. The database is partitioned in subdirectories. Some of these are special: the subdirectory Dirac contains input files for the program *dirac* (computation of relativistic potentials and charge densities). Most subdirectories contain files for the create runs: subdirectories SZ through TZ2P.

The names of the files in the database consist of two parts: the standard symbol for the chemical element and the level of frozen core approximation. Mn.2p for instance is a data file for Manganese with a frozen core up to and including the 2p shell.

Polarization functions are not provided for all elements because our experience with them is limited. If you contemplate to compile more extended basis sets, by including one or more polarization functions, a good rule of thumb to choose the functional characteristics, is the following. Take the next higher *l*-value that does not yet occur in the function set (however, do not go beyond *f*-functions: the program cannot (yet) handle *g*-type basis functions), select the minimum value for the main quantum number *n* that is compatible with the *l*-value (i.e.: 2p, 3d, 4f), and determine the exponential decay factor such that the function attains its maximum value at somewhere between 1/3 and 1/2 times the bond length; the functional maximum for a Slater-type exponential function is at R=(*n*-1)/ a.

A few all-electron basis are included in the data base. In addition, some all-electron sets are provided in the subdirectory Special/ae of the database. However, the files in Special/AE do NOT contain fit functions so they cannot be used directly in Create runs. Fit functions for the all-electron basis sets must include more, in particular more contracted functions that the standard fit sets that are provided in the normal database files. If you would combine the all-electron basis set with an inadequate fit set the results are unreliable and absolutely in adequate, in the same fashion as when you would have used a highly inadequate basis set.

Data File for Create

The data file supplied to adf in Create mode contains the following sections:

```
Title
Basis Functions
Core Expansion Functions
Core Description
Fit Functions
Start-up Fit Coefficients
```

Each of these items is discussed below. The data file does *not* define the applied density functional, the electronic configuration, precision parameters (numerical integration, SCF convergence criterion...), etc etera. These items can be set in the normal input file if the default is not satisfactory.

Title

is the first record of the file. It may contain any text. Only the first 60 characters are actually used. This title is (by default) printed in the output; it is also used to stamp an identification on the result file (TAPE21). The file stamp will be printed whenever you use it as a fragment file in another calculation.

Basis functions

A list of Slater type basis function characteristics. This part has the following format (example):

```
BASIS
  1s 5.4
  2s 1.24
  ...
  (etc.)
  ...
end
```

The words basis and end signal the beginning and the end of this section in the data file. The records in-between list the basis functions; each record contains the main quantum number, the angular quantum number, and the exponential decay factor for a *set* of Slater type basis functions. A function description *3d 2.5* for instance represents the functions *re*Y, *m*=-2,...,2.

The order of specification of the basis functions is not free. First must come the Core Functions used for core-orthogonalization, see Chapter 1.2. The CFs must be in order: s-functions first, then p-functions, then d-functions, and finally f-functions (as far as applicable). In the valence basis set there must be exactly one core-orthogonalization function for each frozen core shell (1s, 2s, 2p, ...).

Here as well as in all other function definitions below, the unit of length, implicit in the exponential decay factor, is bohr (atomic units), irrespective of the unit of length used in input for geometric items such as atomic positions (see units).

Core expansion functions

This part has the form

```
CORE ns, np, nd, nf
  1s 7.68
  ...
  (etc.)
  ...
end
```

It looks very much like the *basis functions*: a list of Slater type function descriptions, closed by end. The header record however (core...) contains in addition four integers ns, np, nd, nf. They are the numbers respectively of *s-, p-, d-, and f-* frozen core shells in the atom. If you create for instance a Ruthenium atom with a frozen core up to the 4p shell, these numbers would be *4 3 1 0* : four frozen *s*-shells (1s,2s,3s,4s), three frozen *p*-shells (2p,3p,4p), one frozen *d*-shell (3d), and no frozen *f*-shells.

The core expansion sets defined in this section are used to describe the frozen core orbitals; they are not included in the valence basis set. In the list of core expansion sets all *s*-type functions must come first, then the *p*-type functions, then the *d*-functions, and then the *f*-functions (as far as applicable).

Core description

Describes the frozen core shells as linear combinations of the core expansion functions. This section has the form

```
COREDESCRIPTION
```

```
  coefficients for the first frozen s-shell
  for the second s-shell
  ...
  for the n-th shell
  coefficients for the first frozen p-shell
  for the second p-shell
  ...
  for the d-shells
  for the f-shells
  pseudopotential parameters
end
```

For each of the angular momentum quantum numbers *l*=*s, p, d, f* all n*l* frozen shells are described by giving expansion coefficients. There are as many coefficients as there are function *sets* with the pertaining *l*-value in the list of expansion functions. There are no separate coefficients for all *m*-values: all *m*-values are equivalent in a spherically symmetric model atom. See the Ca example below.

At the end of the (core) description section there is a record with pseudopotential parameters. The pseudopotential option, as an alternative to the frozen core approximation, is currently not supported, all values in this record must be zero, one for each frozen core shell. Equivalently you can put one zero, followed by a slash (/).

Fit functions

is again a list of Slater type functions. These are used for an expansion of the density. The Coulomb potential due to the electronic charge distribution is computed from this expansion, see Chapter 1.2.

The format of this section is similar to the *basis functions*:

```
FIT
  1s 10.8
  ...
  ...
  (etc.)
  ...
end
```

The program cannot handle fit functions with *l*-value higher than 4, i.e. not higher than *g*-type functions. Bear this in mind if you construct alternative fit sets.

In view of the next item, one is well advised to put the *s*-functions first.

Start-up fit coefficients

The initial (start-up for the SCF procedure) expansion of the atomic charge density in terms of the fit functions. Since the atom is spherically symmetric, only *s*-type functions should have non-zero coefficients. This is why the *s*-type fit functions should be listed first: the list of coefficients can then, after the *s*-set, be closed by a slash, rather than putting a long series of zeros.

The higher *l*-values (p, d...) in the fit set play no role in the creation of the basic atom, because it is spherically symmetric. They should not be omitted however as they will be needed when the atom is used as a fragment in a molecule: the charge density around the atom is then not spherically symmetric anymore.

The form of this section is simple:

```
FITCOEFFICIENTS
coefficients
end
```

Example: Calcium

An example may serve to illustrate the format of a Create data file for Ca (DZ, note that compared to the old basis II an extra 3D polarization function is added) (empty records inside and between the various sections are meaningless and ignored):

```
Calcium (II, 2p frozen)

BASIS
1S  15.8
2S   6.9
2P   8.1

3S   2.6
3S   3.9
3P   2.1
3P   3.4
4S   0.8
4S   1.35
4P   1.06

3D   2.000
END

CORE    2  1  0  0
 1S  24.40
 1S  18.25
 2S   7.40
 2S   4.85
 3S   4.00
 3S   2.55
 4S   0.70
 4S   1.05
 4S   1.65
 2P  10.85
 2P   6.45
 3P   1.85
 3P   2.70
 3P   4.00
END

DESCRIPTION
  0.2076143E+00  0.7975138E+00 -0.7426673E-04  0.1302616E-03 -0.6095738E-04
  0.1508446E-04  0.1549420E-06 -0.2503155E-07 -0.1843317E-05
  0.8487466E-01 -0.4505954E+00  0.1009184E+01  0.9627952E-01 -0.3093986E-01
  0.1678301E-01 -0.2381843E-02  0.6270439E-02 -0.8899688E-02
  0.3454503E+00  0.6922138E+00 -0.1610756E-02  0.5640782E-02 -0.5674517E-02

0/
END
```

```
FIT
 1S  31.80
 2S  29.37
 3S  25.15
 4S  21.06
 4S  13.99
 5S  11.64
 5S   8.05
 6S   6.69
 6S   4.76
 6S   3.39
 7S   2.82
 7S   2.06
 7S   1.50
 2P  24.10
 3P  14.78
 4P   9.29
 5P   5.98
 6P   3.94
 6P   2.24
 7P   1.50
 3D  16.20
 4D  10.47
 5D   6.91
 6D   4.65
 6D   2.70
 7D   1.85
 4F   7.00
 5F   4.00
  5G   3.50
END

FITCOEFFICIENTS
   .56749726864881147OE+02  -.45237728189936717 6E+03   .32614515908773603 3E+03
   .33776564470394245 3E+05   .13130032446710952 2E+04  -.70490321855952634 0E+04
   .75521058772805258 7E+03   .28124173815673117 4E+03   .86492818563053202 0E+01
  -.23002505687873928 1E+00   .36663901111402968 9E-01   .90566300101096184 1E-03
   .16008083216854753 0E-04   .00000000000000000 0E+00   .00000000000000000 0E+00
/
END
```

6.2 Elements of the Periodic Table

A few characteristics are predefined in adf for all elements of the periodic table, as shown below.

The electronic configuration defines the default occupation numbers in Create mode. Basis sets for the elements Rf-Uuo (Z=104-118) are only available in the ZORA atomic database.

	Nuclear Charge Z	*mass number of default isotope used for mass*	*electronic configuration*
H	1	1	$1s^1$
He	2	4	$1s^2$
Li	3	7	$2s^1$
Be	4	9	$2s^2$
B	5	11	$2s^2 2p^1$
C	6	12	$2s^2 2p^2$
N	7	14	$2s^2 2p^3$
O	8	16	$2s^2 2p^4$
F	9	19	$2s^2 2p^5$
Ne	10	20	$2s^2 2p^6$
Na	11	23	$3s^1$
Mg	12	24	$3s^2$
Al	13	27	$3s^2 3p^1$
Si	14	28	$3s^2 3p^2$
P	15	31	$3s^2 3p^3$
S	16	32	$3s^2 3p^4$
Cl	17	35	$3s^2 3p^5$
Ar	18	40	$3s^2 3p^6$
K	19	39	$4s^1$
Ca	20	40	$4s^2$
Sc	21	45	$3d^1 4s^2$
Ti	22	48	$3d^2 4s^2$
V	23	51	$3d^3 4s^2$
Cr	24	52	$3d^5 4s^1$
Mn	25	55	$3d^5 4s^2$
Fe	26	56	$3d^6 4s^2$
Co	27	59	$3d^7 4s^2$
Ni	28	58	$3d^9 4s^1$, $3d^8 4s^2$
Cu	29	63	$3d^{10} 4s^1$
Zn	30	64	$3d^{10} 4s^2$
Ga	31	69	$3d^{10} 4s^2 4p^1$
Ge	32	74	$3d^{10} 4s^2 4p^2$
As	33	75	$3d^{10} 4s^2 4p^3$
Se	34	80	$3d^{10} 4s^2 4p^4$
Br	35	79	$3d^{10} 4s^2 4p^5$

Kr	36	84	$3d^{10}4s^{2}4p^{6}$
Rb	37	85	$5s^{1}$
Sr	38	88	$5s^{2}$
Y	39	89	$4d^{1}5s^{2}$
Zr	40	90	$4d^{2}5s^{2}$
Nb	41	93	$4d^{4}5s^{1}$
Mo	42	98	$4d^{5}5s^{1}$
Tc	43	(98)	$4d^{5}5s^{2}$
Ru	44	102	$4d^{7}5s^{1}$
Rh	45	103	$4d^{8}5s^{1}$
Pd	46	106	$4d^{10}$
Ag	47	107	$4d^{10}5s^{1}$
Cd	48	114	$4d^{10}5s^{2}$
In	49	115	$4d^{10}5s^{2}5p^{1}$
Sn	50	120	$4d^{10}5s^{2}5p^{2}$
Sb	51	121	$4d^{10}5s^{2}5p^{3}$
Te	52	130	$4d^{10}5s^{2}5p^{4}$
I	53	127	$4d^{10}5s^{2}5p^{5}$
Xe	54	132	$4d^{10}5s^{2}5p^{6}$
Cs	55	133	$6s^{1}$
Ba	56	138	$6s^{2}$
La	57	139	$5d^{1}6s^{2}$
Ce	58	140	$4f^{1}5d^{1}6s^{2}$
Pr	59	141	$4f^{3}6s^{2}$
Nd	60	142	$4f^{4}6s^{2}$
Pm	61	(145)	$4f^{5}6s^{2}$
Sm	62	152	$4f^{6}6s^{2}$
Eu	63	153	$4f^{7}6s^{2}$
Gd	64	158	$4f^{7}5d^{1}6s^{2}$
Tb	65	159	$4f^{9}6s^{2}$
Dy	66	164	$4f^{10}6s^{2}$
Ho	67	165	$4f^{11}6s^{2}$
Er	68	166	$4f^{12}6s^{2}$
Tm	69	169	$4f^{13}6s^{2}$
Yb	70	174	$4f^{14}6s^{2}$
Lu	71	175	$4f^{14}5d^{1}6s^{2}$
Hf	72	180	$4f^{14}5d^{2}6s^{2}$
Ta	73	181	$4f^{14}5d^{3}6s^{2}$
W	74	184	$4f^{14}5d^{4}6s^{2}$
Re	75	187	$4f^{14}5d^{5}6s^{2}$
Os	76	192	$4f^{14}5d^{6}6s^{2}$
Ir	77	193	$4f^{14}5d^{7}6s^{2}$
Pt	78	195	$4f^{14}5d^{9}6s^{1}$
Au	79	197	$4f^{14}5d^{10}6s^{1}$

Hg	80	202	$4f^{14}5d^{10}6s^2$
Tl	81	205	$4f^{14}5d^{10}6s^26p^1$
Pb	82	208	$4f^{14}5d^{10}6s^26p^2$
Bi	83	209	$4f^{14}5d^{10}6s^26p^3$
Po	84	(209)	$4f^{14}5d^{10}6s^26p^4$
At	85	(210)	$4f^{14}5d^{10}6s^26p^5$
Rn	86	(222)	$4f^{14}5d^{10}6s^26p^6$
Fr	87	(223)	$7s^1$
Ra	88	(226)	$7s^2$
Ac	89	(227)	$6d^17s^2$
Th	90	232	$6d^27s^2$
Pa	91	231	$5f^26d^17s^2$
U	92	238	$5f^36d^17s^2$
Np	93	(237)	$5f^46d^17s^2$
Pu	94	(244)	$5f^67s^2$
Am	95	(243)	$5f^77s^2$
Cm	96	(247)	$5f^76d^17s^2$
Bk	97	(247)	$5f^97s^2$
Cf	98	(251)	$5f^{10}7s^2$
Es	99	(252)	$5f^{11}7s^2$
Fm	100	(257)	$5f^{12}7s^2$
Md	101	(258)	$5f^{13}7s^2$
No	102	(259)	$5f^{14}7s^2$
Lr	103	(260)	$5f^{14}6d^17s^2$
Rf	104	(261)	$5f^{14}6d^27s^2$
Db	105	(262)	$5f^{14}6d^37s^2$
Sg	106	(263)	$5f^{14}6d^47s^2$
Bh	107	(264)	$5f^{14}6d^57s^2$
Hs	108	(265)	$5f^{14}6d^67s^2$
Mt	109	(268)	$5f^{14}6d^77s^2$
Ds	110	(269)	$5f^{14}6d^87s^2$
Uuu	111	(272)	$5f^{14}6d^97s^2$
Uub	112	(277)	$5f^{14}6d^{10}7s^2$
Uut	113	(280)	$5f^{14}6d^{10}7s^27p^1$
Uuq	114	(280)	$5f^{14}6d^{10}7s^27p^2$
Uup	115	(280)	$5f^{14}6d^{10}7s^27p^3$
Uuh	116	(280)	$5f^{14}6d^{10}7s^27p^4$
Uus	117	(280)	$5f^{14}6d^{10}7s^27p^5$
Uuo	118	(280)	$5f^{14}6d^{10}7s^27p^6$
Default (most abundant) isotope, used to set atomic mass (nr. of brackets gives mass directly). Default electronic configurations used in Create mode.			

6.3 Symmetry

Schönfliess symbols and symmetry labels

A survey of all point groups that are recognized by adf is given below. The table contains the Schönfliess symbols together with the names of the subspecies of the irreducible representations as they are used internally by adf. The subspecies names depend on whether single-group or double-group symmetry is used. Double-group symmetry is used only in relativistic spin-orbit calculations.

Note that for some input of TDDFT (Response) calculations, other conventions apply for the subspecies. This is explicitly mentioned in the discussion of that application.

Point Group	*Schönfliess Symbol in adf*	*Irreducible representations in single-group symmetry*	*Irreducible representations in double-group symmetry*
C_1	NOSYM	A	A1/2
R^3	ATOM	s p d f	s1/2 p1/2 p3/2 d3/2 d5/2 f5/2 f7/2
T_d	T(D)	A1 A2 E T1 T2	E1/2 U3/2 E5/2
O_h	O(H)	A1.g A2.g E.g T1.g T2.g A1.u A2.u E.u T1.u T2.u	E1/2.g U3/2.g E5/2.g E1/2.u U3/2.u E5/2.u
$C_{\infty v}$	C(LIN)	Sigma Pi Delta Phi	J1/2 J3/2 J5/2 J7/2
$D_{\infty h}$	D(LIN)	Sigma.g Sigma.u Pi.g Pi.u Delta.g Delta.u Phi.g Phi.u	J1/2.g J1/2.u J3/2.g J3/2.u J5/2.g J5/2.u J7/2.g J7/2.u
C_i	C(I)	A.g A.u	A1/2.g A1/2.u
C_s	C(S)	AA AAA	A1/2 A1/2*
C_n	C(N), n must be 2	A B E1 E2 ... odd n: without B	A1/2 A1/2*
C_{nh}	C(NH), n must be 2	even n: A.g B.g A.u B.u E1.g E1.u E2.g E2.u ... odd n: AA AAA EE1 EE2 ... EEE1 EEE2 ...	A1/2.g A1/2.g* A1/2.u A1/2.u*
C_{nv}	C(NV), n<9	A1 A2 B1 B2 E1 E2 E3 ... odd n: without B1 and B2	E1/2 E3/2 E5/2 ... for even n also: An/2 An/2*
D_n	D(N), n<9	n=2: A B1 B2 B3 other: A1 A2 B1 B2 E1 E2 E3 ... odd n: without B1 B2	E1/2 E3/2 ... for odd n also: An/2 An/2*
D_{nh}	D(NH), n<9	n=2: A1.g B1.g B2.g B3.g A1.u B1.u B2.u B3.u even n (≠2): A1.g A2.g B1.g B2.g E1.g E2.g E3.g ... A1.u A2.u B1.u ... odd n: AA1 AA2 EE1 EE2 ... AAA1 AAA2 EEE1 EEE2	even n: E1/2.g E1/2.u E3/2.g E3/2.u ... odd n: E1/2 E3/2 E5/2 ...
D_{nd}	D(ND), n<9	n=2: A1 A2 B1 B2 E1 other: A1.g A2.g E1.g E2.g ... E(n-1)/2.g A1.u A2.u E1.u E2.u ... E(n-1)/2.u	even n: E1/2 E3/2 ... odd n: E1/2.g E1/2.u E3/2.g E3/2.u An/2.g An/2.u An/2.g* An/2.u*

Schönfliess symbols and the labels of the irreducible representations.

Most labels are easily associated with the notation usually encountered in literature. Exceptions are AA, AAA, EE1, EEE1, EE2, EEE2, etc etera. They stand for A', A'', E1', E1'', and so on. The AA, etc. notation is used in adf to avoid using quotes in input files in case the subspecies names must be referred to.

The symmetry labeling of orbitals may depend on the choice of coordinate system. For instance, B1 and B2 representations in Care interchanged when you rotate the system by 90 degrees around the z-axis so that

x-axis becomes y-axis and vice-versa (apart from sign).

Labels of the symmetry subspecies are easily derived from those for the irreps. For one-dimensional representations they are identical, for more-dimensional representations a suffix is added, separated by a colon:
For the two- and three-dimensional E and T representations the subspecies labels are obtained by adding simply a counting index (1, 2, 3) to the name, with a colon in between; for instance, the EE1 irrep in the Dnh pointgroup has EE1:1 and EE1:2 subspecies. The same holds for the two-dimensional representations of $C_{\infty v}$ and $D_{\infty h}$. For the R3 (atom) point group symmetry the subspecies are p:x, p:y, p:z, d:z2, d:x2-y2, etc.

All subspecies labels are listed in the Symmetry section, very early in the ADF output. To get this, perform a quick run of the molecule using the STOPAFTER key (for instance: stopafter config).

Molecular orientation requirements

adf requires that the molecule has a specific orientation in space, as follows:

- The origin is a fixed point of the symmetry group.
- The z-axis is the main rotation axis, xy is the σ_h-plane (axial groups, C(s)).
- The x-axis is a C_2 axis (D symmetries).
- The xz-plane is a σ_v-plane (C_{nv} symmetries).
- In T_d and O_h the z-axis is a fourfold axis (S_4 and C_4, respectively) and the (111)-direction is a threefold axis.

If the user-specified symmetry equals the true symmetry of the nuclear frame (including electric field and point charges) the program will adapt the input coordinates to the above requirements, if necessary. If no symmetry has been specified at all adf assumes you have specified the symmetry of the nuclear frame, accounting for any fields. If a subgroup has been specified for the molecular symmetry the input coordinates will be used as specified by the user. If a Z-matrix input is given this implies for the Cartesian coordinates: first atom in the origin, second atom on the positive x-axis, third atom in the xy-plane with positive y value.

7 References

1. Baerends, E.J., V. Branchadell, and M. Sodupe, *Atomic reference energies for density functional calculations.* Chemical Physics Letters, 1997. **265**: p. 481.

2. Bickelhaupt, F.M. and E.J. Baerends, *Kohn-Sham DFT: Predicting and Understanding Chemistry*, in *Reviews of Computational Chemistry*, D.B. Boyd and K.B. Lipkowitz, Editors. 2000, Wiley-VCH: New York. p. 1-86.

3. Ziegler, T. and A. Rauk, *On the calculation of Bonding Energies by the Hartree Fock Slater method. I. The Transition State Method.* Theoretica Chimica Acta, 1978. **46**: p. 1-10.

4. van den Hoek, P.J., A.W. Kleyn, and E.J. Baerends, *What is the origin of the repulsive wall in atom-atom potentials.* Comments Atomic and Molecular Physics, 1989. **23**: p. 93.

5. Baerends, E.J., *Pauli repulsion effects in scattering from and catalysis by surface*, in *Cluster models for surface and bulk phenomena*, G. Puccchiori, P.S. Bagus, and F. Parmigiani, Editors. in press. p. 189-207.

6. Versluis, L. and T. Ziegler, *The determination of Molecular Structure by Density Functional Theory.* Journal of Chemical Physics, 1988. **88**: p. 322.

7. Versluis, L., *The determination of molecular structures by the HFS method*. 1989, University of Calgary.

8. Fan, L. and T. Ziegler, *Optimization of molecular structures by self consistent and non-local density functional theory.* Journal of Chemical Physics, 1991. **95**: p. 7401.

9. Deng, L. and T. Ziegler, *A combined density functional and intrinsic reaction coordinate study on the ground state energy surface of H_2CO.* Journal of Chemical Physics, 1993. **99**: p. 3823.

10. Deng, L. and T. Ziegler, *The determination of Intrinsic Reaction Coordinates by density functional theory.* International Journal of Quantum Chemistry, 1994. **52**: p. 731-765.

11. Fischer, T.H. and J. Almlöf, *General Methods for Geometry and Wave Function Optimization.* Journal of Physical Chemistry, 1992. **96**: p. 9768-9774.

12. Fan, L. and T. Ziegler, *Application of density functional theory to infrared absorption intensity calculations on main group molecules.* Journal of Chemical Physics, 1992. **96**: p. 9005-9012.

13. Fan, L. and T. Ziegler, *Nonlocal density functional theory as a practical tool in calculations on transition states and activation energies.* Journal of the American Chemical Society, 1992. **114**: p. 10890.

14. Bérces, A., *Application of density functional theory to the vibrational characterization of transition metal complexes*. 1995, University of Calgary: Calgary.

15. van Leeuwen, R. and E.J. Baerends, *Exchange-correlation potential with correct asymptotic behavior.* Physical Review A, 1994. **49**(4): p. 2421-2431.

16. Grüning, M., O.V.Gritsenko, S.J.A. van Gisbergen and E.J. Baerends, *Shape corrections to exchange-correlation Kohn-Sham potentials by gradient-regulated seamless connection of model potentials for inner and outer region.* Journal of Chemical Physics, 2001. **114**: p. 652-660.

17. Schipper, P.R.T., O.V. Gritsenko, S.J.A. van Gisbergen, and E.J. Baerends, *Molecular calculations of excitation energies and (hyper)polarizabilities with a statistical average of orbital model exchange-correlation potentials.* Journal of Chemical Physics, 2000. **112**: p. 1344-1352.

18. Grüning M., O.V. Gritsenko, S.J.A. van Gisbergen, E.J. Baerends, *On the required shape correction to the LDA and GGA Kohn Sham potentials for molecular response calculations of (hyper)polarizabilities and excitation energies.* Journal of Chemical Physics, 2002. **116**(22): p. 9591-9601.

19. Chong, D.P., O.V. Gritsenko, and E.J. Baerends, Journal of Chemical Physics, 2002. **116**: p. 1760.

20. Vosko, S.H., L. Wilk, and M. Nusair, *Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis.* Canadian Journal of Physics, 1980. **58**: p. 1200.

21. Stoll, H., C.M.E. Pavlidou, and H. Preuss, *On the calculation of correlation energies in the spin-density functional formalism.* Theoretica Chimica Acta, 1978. **49**: p. 143.

22. Becke, A.D., Physical Review A, 1988. **38**: p. 3098.

23. Perdew, J.P. and Y. Wang, *Accurate and simple density functional for the electronic exchange energy: generalized gradient approximation.* Physical Review B, 1986. **33**(12): p. 8822.

24. Perdew, J.P., J.A. Chevary, S.H. Vosko, K.A. Jackson, M.R. Pederson, D.J. Singh, C. Fiolhais, Physical Review B, 1992. **46**: p. 6671.

25. Adamo, C. and V. Barone, Journal of Chemical Physics, 1998. **108**: p. 664.

26. Perdew, J.P., K. Burke, and M. Ernzerhof, Physical Review Letters, 1996. **77**: p. 3865.

27. Hammer, B., L.B. Hansen, and J.K. Norskov, Physical Review, 1999. **B59**: p. 7413.

28. Zhang, Y. and W. Yang, Physical Review Letters, 1998. **80**: p. 890.

29. Handy, N.C. and A.J. Cohen, Molecular Physics, 2001. **99**: p. 403.

30. Perdew, J.P., *Density-functional approximation for the correlation energy of the inhomogeneous electron gas.* Physical Review B, 1986. **33**(12): p. 8822.
Erratum: J.P. Perdew, Phys. Rev. B, 1986**34** p. 7406.

31. Lee, C., W. Yang, and R.G. Parr, *Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density.* Physical Review B, 1988. **37**(2): p. 785.

32. Johnson, B.G., P.M.W. Gill, and J.A. Pople, *The performance of a family of density functional methods.* Journal of Chemical Physics, 1993. **98**(7): p. 5612.

33. Russo, T.V., R.L. Martin, and P.J. Hay, *Density Functional calculations on first-row transition metals.* Journal of Chemical Physics, 1994. **101**(9): p. 7729.

34. Neumann, R., R.H. Nobes, and N.C. Handy, *Exchange functionals and potentials.* Molecular Physics, 1996. **87**(1): p. 1-36.

35. Perdew, J.P., K. Burke, and M. Ernzerhof, Physical Review Letters, 1997. **78**: p. 1396.

36. Hamprecht, F.A., A.J. Cohen, D.J. Tozer, and N.C. Handy, Journal of Chemical Physics, 1998. **109**: p. 6264.

37. Boese, A.D., N.L. Doltsinis, N.C. Handy, and M. Sprik, Journal of Chemical Physics, 2000. **112**: p. 1670.

38. Boese, A.D. and N.C. Handy, *A new parametrization of exchange-correlation generalized gradient approximation functionals.* Journal of Chemical Physics, 2001. **114**: p. 5497-5503.

39. Tsuneda, T., T. Suzumura, and K. Hirao, Journal of Chemical Physics, 1999. **110**: p. 10664.

40. Krieger, J.B., J. Chen, G.J. Iafrate, and A. Savin, in *Electron Correlations and Materials Properties*, A. Gonis and N. Kioussis, Editors. 1999, Plenum: New York.

41. Perdew, J.P., S. Kurth, A. Zupan, and P. Blaha, Physical Review Letters, 1999. **82**: p. 5179.

42. van Voorhis, T. and G.E. Scuseria, Journal of Chemical Physics, 1998. **109**: p. 400.

43. Filatov, M. and W. Thiel, Molecular Physics, 1997. **91**: p. 847.

44. Filatov, M. and W. Thiel, Physical Review A, 1998. **57**: p. 189.

45. Proynov, E.I., S. Sirois, and D.R. Salahub, International Journal of Quantum Chemistry, 1997. **64**: p. 427.

46. Proynov, E.I., H. Chermette, and D.R. Salahub, Journal of Chemical Physics, 2000. **113**: p. 10013.

47. Patchkovskii, S., J. Autschbach, and T. Ziegler, *Curing difficult cases in magnetic properties prediction with self-interaction corrected density functional theory.* Journal of Chemical Physics, 2001. **115**: p. 26-42.

48. Patchkovskii, S. and T. Ziegler, *Improving `difficult' reaction barriers with self-interaction corrected density functional theory.* Journal of Chemical Physics, 2002. **116**: p. 7806.

49. Patchkovskii, S. and T. Ziegler, *"Phosphorus NMR chemical shifts with self-interaction free, gradient-corrected DFT.* Journal of Physical Chemistry, 2002. **A 106**: p. 1088.

50. Philipsen, P.H.T., E. van Lenthe, J.G. Snijders and E.J. Baerends, *Relativistic calculations on the adsorption of CO on the (111) surfaces of Ni, Pd, and Pt within the zeroth-order regular approximation.* Physical Review B, 1997. **56**(20): p. 13556-13562.

51. Snijders, J.G. and E.J. Baerends, *A perturbation theory approach to relativistic calculations. I. Atoms.* Molecular Physics, 1978. **36**: p. 1789.

52. Snijders, J.G., E.J. Baerends, and P. Ros, *A perturbation theory approach to relativistic calculations. II. Molecules.* Molecular Physics, 1979. **38**: p. 1909.

53. Ziegler, T., J.G. Snijders, and E.J. Baerends, *Relativistic effects on bonding.* Journal of Chemical Physics, 1981. **74**: p. 1271.

54. DeKock, R.L., E.J. Baerends, P.M. Boerrigter and J.G. Snijders, *On the nature of the first excited states of the uranyl ion.* Chemical Physics Letters, 1984. **105**: p. 308.

55. DeKock, R.L., E.J. Baerends, P.M. Boerrigter and R. Hengelmolen, *Electronic structure and bonding of $Hg(CH_3)_2$, $Hg(CN)_2$, $Hg(CH_3)(CN)$, $Hg(CCCH_3)_2$, and $Au(PMe)_3)(CH_3)$.* Journal of the American Chemical Society, 1984. **106**: p. 33387.

56. Boerrigter, P.M., *Spectroscopy and bonding of heavy element compounds*. 1987, Vrije Universiteit.

57. Boerrigter, P.M., M.A. Buijse, and J.G. Snijders, *Spin-Orbit interaction in the excited states of the dihalogen ions F2+, Cl2+ an Br2+.* Chemical Physics, 1987. **111**: p. 47-53.

58. Boerrigter, P.M., E.J. Baerends, and J.G. Snijders, *A relativistic LCAO Hartree-Fock-Slater investigation of the electronic structure of the actinocenes $M(COT)_2$, M=Th, Pa, U, Np and Pu.* Chemical Physics, 1988. **122**: p. 357-374.

59. Ziegler, T., V. Tschinke, E.J. Baerends, J.G. Snijders and W. Ravenek, *Calculation of bond energies in compounds of heavy elements by a quasi-relativistic approach.* Journal of Physical Chemistry, 1989. **93**: p. 3050-3056.

60. Li, J., G. Schreckenbach, and T. Ziegler, Journal of the American Chemical Society, 1995. **117**: p. 486.

61. van Lenthe, E., A.E. Ehlers, and E.J. Baerends, *Geometry optimization in the Zero Order Regular Approximation for relativistic effects.* Journal of Chemical Physics, 1999. **110**: p. 8943-8953.

62. van Lenthe, E., E.J. Baerends, and J.G. Snijders, *Relativistic regular two-component Hamiltonians.* Journal of Chemical Physics, 1993. **99**: p. 4597.

63. van Lenthe, E., E.J. Baerends, and J.G. Snijders, *Relativistic total energy using regular approximations.* Journal of Chemical Physics, 1994. **101**(11): p. 9783.

64. van Lenthe, E., J.G. Snijders, and E.J. Baerends, *The zero-order regular approximation for relativistic effects: The effect of spin-orbit coupling in closed shell molecules.* J. Chem. Phys., 1996. **105**(15): p. 6505-6516.

65. van Lenthe, E., R. van Leeuwen, E.J. Baerends, and J.G. Snijders, *Relativistic regular two-component Hamiltonians.* International Journal of Quantum Chemistry, 1996. **57**: p. 281 -293.

66. Pye, C.C. and T. Ziegler, Theoretical Chemistry Accounts, 1999. **101**: p. 396-408.

67. Klamt, A. and G. Schüürmann, *COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient.* Journal of the Chemical Society: Perkin Transactions, 1993. **2**: p. 799-805.

68. Klamt, A., *Conductor-like Screening Model for real solvents: A new approach to the quantitative calculation of solvation phenomena.* Journal of Physical Chemistry, 1995. **99**: p. 2224-2235.

69. Klamt, A. and V. Jones, *Treatment of the outlying charge in continuum solvation models.* Journal of Chemical Physics, 1996. **105**: p. 9972-9981.

70. Pascual-Ahuir, J.L., E. Silla, and I. Tunon, Journal of Computational Chemistry, 1994. **15**: p. 1127.

71. van Gisbergen, S.J.A., J.G. Snijders, and E.J. Baerends, Computer Physics Communications, 1999. **118**: p. 119-138.

72. van Gisbergen, S.J.A., *Molecular Response Property Calculations using Time Dependent Density Functional Theory*, in *Chemistry*. 1998, Vrije Universiteit: Amsterdam. p. 190.

73. Gross, E.K.U., J.F. Dobson, and Petersilka, in *Density Functional Theory*, R.F. Nalewajski, Editor. 1996, Springer: Heidelberg.

74. van Gisbergen, S.J.A., V.P. Osinga, O.V. Gritsenko, R. van Leeuwen, J.G. Snijders and E.J. Baerends, *Improved density functional theory results for frequency-dependent polarizabilities, by the use of an exchange-correlation potential with correct asymptotic behavior.* Journal of Chemical Physics, 1996. **105**(7): p. 3142.

75. van Gisbergen, S.J.A., J.G. Snijders, and E.J. Baerends, Physical Review Letters, 1997. **78**: p. 3097.

76. van Gisbergen, S.J.A., J.G. Snijders, and E.J. Baerends, *Calculating frequency-dependent hyperpolarizabilities using time-dependent density functional theory.* Journal of Chemical Physics, 1998. **109**: p. 10644-10656.

77. van Gisbergen, S.J.A., J.G. Snijders, and E.J. Baerends, *Accurate density functional calculations on frequency-dependent hyperpolarizabilities of small molecules.* Journal of Chemical Physics, 1998. **109**: p. 10657-10668.

78. Casida, M.E., C. Jamorski, K.C. Casida, and D.R. Salahub, Journal of Chemical Physics, 1998. **108**: p. 4439.

79. Osinga, V.P., S.J.A. van Gisbergen, J.G. Snijders and E.J. Baerends, Journal of Chemical Physics, 1997. **106**: p. 5091.

80. Autschbach, J. and T. Ziegler, *Calculating molecular electric and magnetic properties from time-dependent density functional response theory.* Journal of Chemical Physics, 2002. **116**: p. 891-896.

81. Autschbach, J., T. Ziegler, S.J.A. van Gisbergen, and E.J. Baerends, *CD.* Journal of Chemical Physics, 2002. **116**: p. 6930.

82. van Gisbergen, S.J.A., F. Kootstra, P.R.T. Schipper, O.V. Gritsenko, J.G. Snijders & E.J. Baerends, Physical Review A, 1998. **57**: p. 2556.

83. van Gisbergen, S.J.A., A. Rosa, G. Ricciardi, and E.J. Baerends, Journal of Chemical Physics, 1999. **111**: p. 2499.

84. Rosa, A., G. Ricciardi, E.J. Baerends, S.J.A. van Gisbergen, *The Optical Spectra of NiP, Nipz, NiTBP, and NiPc. Electronic effects of meso-tetraaza substitution and tetrabenzoannulation.* Journal of Physical Chemistry A, 2001. **105**: p. 3311-3327.

85. Ricciardi, G., A. Rosa, and E.J. Baerends, *Ground and Excited States of Zinc Phthalocyanine studied by Density Functional Methods.* Journal of Physical Chemistry A, 2001. **105**: p. 5242-5254.

86. van Gisbergen, S.J.A., J.A. Groeneveld, A. Rosa, J.G. Snijders and E.J.Baerends, *Excitation energies for transition metal compounds from time-dependent density functional theory. Applications to MnO4-, Ni(CO)4 and Mn2(CO)10.* Journal of Physical Chemistry A, 1999. **103**: p. 6835-6844.

87. Rosa, A., E.J. Baerends, S.J.A. van Gisbergen, E. van Lenthe, J.A. Groeneveld and J. G. Snijders, Journal of the American Chemical Society, 1999. **121**: p. 10356-10365.

88. van Gisbergen, S.J.A., C. Fonseca Guerra, and E.J. Baerends, *Towards excitation energies and (hyper)polarizability calculations of large molecules. Application of parallelization and linear scaling techniques to time-dependent density functional response theory.* Journal of Computational Chemistry, 2000. **21**: p. 1511-1523.

89. van Gisbergen, S.J.A., J.G. Snijders, and E.J. Baerends, *A Density Functional Theory study of frequency-dependent polarizabilities and van der Waals dispersion coefficients for polyatomic molecules.* Journal of Chemical Physics, 1995. **103**: p. 9347-9354.

90. Champagne, B., E.A. Perpète, S.J.A. van Gisbergen, E.J. Baerends, J.G. Snijders, C. Soubra-Ghaoui, K.A. Robins & B.Kirtman, *Assessment of conventional density functional schemes for computing the polarizabilities and hyperpolarizabilities of conjugated oligomers: An ab initio investigation of polyacetylene chains.* Journal of Chemical Physics, 1998. **109**: p. 10489-10498. Erratum: J. Chem. Phys., 1999 **111**: 6652

91. Bishop, D.M., Advances in Quantum Chemistry, 1994. **25**: p. 3.

92. Willets, A., J.E. Rice, D.M. Burland, and D.P. Shelton, *Problems in comparison of experimental and theoretical hyperpolarizabilities.* Journal of Chemical Physics, 1992. **97**: p. 7590.

93. Shelton, D.P. and J.E. Rice, Chemical Reviews, 1994. **94**: p. 3.

94. Autschbach, J., S. Patchkovskii, T. Ziegler, S.J.A. van Gisbergen, and E.J. Baerends, *ORD.* Journal of Chemical Physics, 2002. **117**: p. 581.

95. van Lenthe, E., A. van der Avoird, and P.E.S. Wormer, *Density functional calculations of molecular g-tensors in the zero order regular approximation for relativistic effects.* Journal of Chemical Physics, 1997. **107**: p. 2488-2498.

96. van Lenthe, E., A. van der Avoird, and P.E.S. Wormer, *Density functional calculations of molecular hyperfine interactions in the zero order regular approximation for relativistic effects.* Journal of Chemical Physics, 1998. **108**(12): p. 4783-4796.

97. van Lenthe, E. and E.J. Baerends, *Density functional calculations of nuclear quadrupole coupling constants in the zero-order regular approximation for relativistic effects.* Journal of Chemical Physics, 2000. **112**(19): p. 8279-8292.

98. Bulo, R.E., A.W. Ehlers, S. Grimme, and K. Lammertsma, Journal of the American Chemical Society, 2002. **124**: p. 13903.

99. Pauncz, R., *Spin Eigenfunctions*. 1979, New York: Plenum Press.

100. Szabo, A. and N.S. Ostlund, *Modern Quantum Chemistry*. 1st ed. revised ed. 1989: McGraw-Hill.

101. Eschrig, H. and V.D.P. Servedio, *Relativistic density functional approach to open shells.* Journal of Computational Chemistry, 1999. **20**: p. 23.

102. Wullen, C.v., *Spin densities in two-component relativistic density functional calculations: Noncollinear versus collinear approach.* Journal of Computational Chemistry, 2002. **23**: p. 779.

103. Wang, S.G. and W.H.E. Schwarz, *Simulation of nondynamical correlation in density functional calculations by the optimized fractional orbital occupation approach: Application to the potential energy surfaces of O_3 and SO_2.* Journal of Chemical Physics, 1996. **105**(11): p. 4641.

104. Boerrigter, P.M., G. te Velde, and E.J. Baerends, *Three-dimensional Numerical Integration for Electronic Structure Calculations.* International Journal of Quantum Chemistry, 1988. **33**: p. 87.

105. te Velde, G. and E.J. Baerends, *Numerical Integration for Polyatomic Systems.* Journal of Computational Physics, 1992. **99**(1): p. 84-98.

106. Bérces, A. and T. Ziegler, Chemical Physics Letters, 1993. **203**(5-6): p. 592.

107. Bérces, A. and T. Ziegler, Journal of Chemical Physics, 1993. **98**: p. 4793.

108. Bérces, A. and T. Ziegler, *The harmonic force field of benzene. A local density functional study.* Journal of Chemical Physics, 1993. **98**: p. 4793.

109. te Velde, G., *Numerical integration and other methodological aspects of bandstructure calculations*, in *Chemistry*. 1990, Vrije Universiteit: Amsterdam.

110. Ziegler, T. and A. Rauk, *On the calculation of Bonding Energies by the Hartree Fock Slater method.* Theoretica Chimica Acta, 1977. **49**: p. 143.

111. Noodleman, L. and E.J. Baerends, *Electronic Structure, Magnetic Properties, ESR, and Optical Spectra for 2-Fe Ferredoxin Models by LCAO-Xa Valence Bond Theory.* Journal of the American Chemical Society, 1984. **106**: p. 2316.

112. Bickelhaupt, F.M., N.M. Nibbering, E.M. van Wezenbeek and E.J. Baerends, *The Central Bond in the Three CN* Dimers NC_CN, CN-CN, and CN-NC: Electron Pair Bonding and Pauli Repulsion Effects.* Journal of Physical Chemistry, 1992. **96**(12): p. 4864.

113. Schreckenbach, G. and T. Ziegler, *The calculation of NMR shielding tensors using GIAO's and modern density functional theory.* Journal of Physical Chemistry, 1995. **99**: p. 606.

114. Schreckenbach, G. and T. Ziegler, *The calculation of NMR shielding tensors based on density functional theory and the frozen-core approximation.* International Journal of Quantum Chemistry, 1996. **60**: p. 753-766.

115. Schreckenbach, G. and T. Ziegler, *Calculation of NMR shielding tensors based on density functional theory and a scalar relativistic Pauli-type Hamiltonian. The application to transition metal complexes.* International Journal of Quantum Chemistry, 1997. **61**: p. 899.

116. Wolff, S.K. and T. Ziegler, *Calculation of DFT-GIAO NMR shifts with inclusion of spin-orbit coupling.* Journal of Chemical Physics, 1998. **109**: p. 895.

117. Wolff, S.K., T. Ziegler, E. van Lenthe, and E.J. Baerends, *Density functional calculations of nuclear magnetic shieldings using the zeroth-order regular approximation for relativistic effects: ZORA NMR.* Journal of Chemical Physics, 1999. **110**: p. 7689.

118. Autschbach, J. and T. Ziegler, *Nuclear spin-spin coupling constants from regular approximate density functional calculations. I. Formalism and scalar relativistic results for heavy metal compounds.* Journal of Chemical Physics, 2000. **113**: p. 936-947.

119. Autschbach, J. and T. Ziegler, *Nuclear spin-spin coupling constants from regular approximate relativistic density functional calculations. II. Spin-orbit coupling effects and anisotropies.* Journal of Chemical Physics, 2000. **113**: p. 9410-9418.

120. Schreckenbach, G. and T. Ziegler, *Calculation of the G-tensor of electron paramagnetic resonance spectroscopy using Gauge-Including Atomic Orbitals and Density Functional Theory.* Journal of Physical Chemistry A, 1997. **101**: p. 3388-3399.

121. Patchkovskii, S. and T. Ziegler, Journal of Physical Chemistry, 2001. **A105**: p. 5490.

122. Edmiston, C. and K. Rüdenberg, Rev. Mod. Phys., 1963. **35**: p. 457.

123. Boys, S.F., Rev. Mod. Phys., 1960. **32**: p. 253.

124. von Niessen, W., Journal of Chemical Physics, 1967. **47**: p. 253.

125. Hirshfeld, F.L., Theoretica Chimica Acta, 1977. **44**: p. 129.

126. Wiberg, K.B. and P.R. Rablen, *Comparison of atomic charges derived via different procedures.* J. Comp. Chem., 1993. **14**(12): p. 1504.

127. Bickelhaupt, N.J.R. van Eikema Hommes, C. Fonseca Guerra, and E.J. Baerends, *The Carbon-Lithium Electron Pair Bond in (CH3Li)n (n = 1, 2, 4).* Organometallics, 1996. **15**: p. 2923.

128. C. Fonseca Guerra, J.-W. Handgraaf, E. J. Baerends, F. M. Bickelhaupt, *Voronoi Deformation Density (VDD) charges. Assessment of the Mulliken, Bader, Hirshfeld, Weinhold and VDD methods for Charge Analysis*, J. Comput. Chem. 2004, **25**, p. 189.

129. Fonseca Guerra, C., F.M. Bickelhaupt, J.G. Snijders, E.J. Baerends, Chemical European Journal, 1999. **5**: p. 3581-3594.

130. Kitaura, K. and K. Morokuma, International Journal of Quantum Chemistry, 1976. **10**: p. 325.

131. Ziegler, T. and A. Rauk, Inorganic Chemistry, 1979. **18**: p. 1755.

132. Fujimoto, H., J. Osamura, and T. Minato, Journal of the American Chemical Society, 1978. **100**: p. 2954.

133. Wolfe, S., D.J. Mitchell, and M.H. Whangbo, Journal of the American Chemical Society, 1978. **100**: p. 1936.

134. Stone, A.J. and R.W. Erskine, Journal of the American Chemical Society, 1980. **102**: p. 7185.

135. Bernardi, F., A. Bottoni, A. Mangini, and G. Tonachini, THEOCHEM-Journal of Molecular Structure, 1981. **3**: p. 163.

136. van den Hoek, P.J. and E.J. Baerends, *Chemical bonding at metal-semiconductor interfaces.* Applied Surface Science, 1989. **41/42**: p. 236.

137. Autschbach, J., *On the calculation of relativistic effects and how to understand their trends in atoms and molecules*, in *Chemistry*. 1999, University of Siegen: Siegen.

138. M.A. Watson, N.C. Handy, and A.J. Cohen, Journal of Chemical Physics, 2003. **119**: p. 6475.

139. Ö. Farkas and H.B. Schlegel, Physical Chemistry Chemical Physics, 2002. **4**: p. 11.

140. I. Mayer, *Charge, bond order and valence in the ab inition SCF theory*, Chemical Physics Letters, 1983. **97**: p. 270.

141. L. Jensen, P.T. van Duijnen, and J.G. Snijders, Journal of Chemical Physics, 2003 **118**: p. 514

142. L. Jensen, P.T. van Duijnen, and J.G. Snijders, Journal of Chemical Physics, 2003 **119**: p. 3800

143. L. Jensen, P.T. van Duijnen, and J.G. Snijders, Journal of Chemical Physics, 2003 **119**: p. 12998

144. L. Jensen, M. Swart, and P.T. van Duijnen, Journal of Chemical Physics, 2005 **122**: p. 034103

145. L. Jensen, *Modelling of optical response properties: Application to nanostructures*, PhD thesis, Rijksuniversiteit Groningen, 2004.

146. P.T. van Duijnen and M. Swart, Journal of Physical Chemistry A, 1998 **102**, p. 2399

147. L. Jensen, P.-O. Astrand, A. Osted, J. Kongsted, and K.V. Mikkelsen, Journal of Chemical Physics, 2002 **116**: p. 4001

148. A. Michalak, R.L. DeKock, and T. Ziegler, J. Comput. Chem., in preparation

149. R.F. Nalewajski, and J. Mrozek, International Journal of Quantum Chemistry, 1994 **51**: p. 187

150. R.F. Nalewajski, J. Mrozek, and A. Michalak, International Journal of Quantum Chemistry, 1997 **61**: p. 589

151. R.F. Nalewajski, J. Mrozek, and A. Michalak, Polish J. Chem., 1998 **72**: p. 1779

152. R.F. Nalewajski, J. Mrozek, and G. Mazur, Can J. Chem., 1996 **74**: p. 1121

153. M.S. Gopinathan, and K. Jug, Theoret. Chim. Acta 1983 **63**: p. 497

154. F. Wang, and T. Ziegler, *Excitation energies of some d1 systems calculated using time-dependent density functional theory: an implementation of open-shell TDDFT theory for doublet-doublet excitations.* Molecular Physics, 2004 **102**: p. 2585

155. Z. Rinkevicius, I. Tunell, P. Salek, O. Vahtras, and H. Agren, *Restricted density functional theory of linear time-dependent properties in open-shell molecules.* Journal of Chemical Physics, 2003 **119**: p. 34

156. F. Wang, and T. Ziegler, *Time-dependent density functional theory based on a noncollinear formulation of the exchange-correlation potential.* Journal of Chemical Physics, 2004 **121**: p. 12191

157. F. Wang, and T. Ziegler, *The performance of time-dependent density functional theory based on a noncollinear exchange-correlation potential in the calculations of excitation energies.* Journal of Chemical Physics, 2005 **122**: p. 074109

158. S. Hirata, and M. Head-Gordon, *Time-dependent density functional theory within the Tamm-Dancoff approximation.* Chemical Physics Letter, 1999 **314**: p. 291

159. G. Henkelman, B.P. Uberuaga, and H. Jonsson, *A climbing image nudged elastic band method for finding saddle points and minimum energy paths.* Journal of Chemical Physics, 2000 **113**: p. 9901

160. G. Vignale and W. Kohn, Phys. Rev. Lett. 1996, **77**, p. 2037

161. G. Vignale and W. Kohn, in *Electronic Density Functional Theory: Recent Progress and New Direction*, J. F. Dobson, G. Vignale, and M. P. Das, Editors. 1998, Plenum: New York.

162. M. van Faassen, P. L. de Boeij, R. van Leeuwen, J. A. Berger, and J. G. Snijders, Phys. Rev. Lett., 2002 **88**: p. 186401

163. M. van Faassen, P. L. de Boeij, R. van Leeuwen, J. A. Berger, and J. G. Snijders, Journal of Chemical Physics, 2003 **118**: p. 1044

164. M. van Faassen and P. L. de Boeij, Journal of Chemical Physics, 2004 **120**: p. 8353

165. M. van Faassen and P. L. de Boeij, Journal of Chemical Physics, 2004 **121**: p. 10707

166. M. van Faassen, *Time-Dependent Current-Density-Functional Theory for Molecules*, PhD thesis, Rijksuniversiteit Groningen, 2004.

167. R. Nifosi, S. Conti, and M. P. Tosi, Phys. Rev. B, 1998 **58**: p. 12758

168. Z. X. Qian and G. Vignale, Phys. Rev. B, 2003 **68**: p. 195113

169. M. Stener, G. Fronzoni, and M. de Simone, Chem. Phys. Lett., 2003 **373**: p. 115

170. M. Swart, P.Th. van Duijnen, J.G. Snijders, J. Comput. Chem., 2001 **22**: p. 79

171. T.W. Keal, and D.J. Tozer, J. Chem. Phys., 2003 **119**: p. 3015

172. X. Xu and W.A. Goddard III, PNAS, 2004 **101**: p. 2673

173. J. Baker, A. Kessi, and B. Delley, J. Chem. Phys., 1996 **105**: p. 192

174. C. Adamo and V. Barone, J. Chem. Phys., 1996 **116**: p. 5933

175. M. Swart, A.W. Ehlers, and K. Lammertsma, Mol. Phys., 2004 **102**: p. 2467

176. P.J. Stephens, F.J. Devlin, C.F. Chabalowski and M.J. Frisch, J. Phys. Chem., 1994 **98**: p. 11623-11627

177. M. Reiher, O. Salomon and B.A. Hess, Theor. Chem. Acc., 2001 **107**: p. 48

178. C. Adamo and V. Barone, Chem. Phys. Lett., 1997 **274**: p. 242

179. J.K. Kang and C.B. Musgrave, J. Chem. Phys., 2001 **115**: p. 11040

180. A.J. Cohen and N.C. Handy, Mol. Phys., 2001, **99**: p. 607-615

181. B.J. Lynch, P.L. Fast, M. Harris and D.G. Truhlar, J. Phys. Chem. A, 2000, **104**: p. 4811

182. F. Wang, T. Ziegler, E. van Lenthe, S.J.A. vand Gisbergen, and E.J. Baerends, *The calculation of excitation energies based on the relativistic two-component zeroth-order regular approximation and time-dependent density-functional with full use of symmetry* Journal of Chemical Physics, 2005 **122**: p. 204103

183. F. Wang, and T. Ziegler, *Theoretical study of the electronic spectra of square-planar platinum (II) complexes based on the two-component relativistic time-dependent density-functional theory* Journal of Chemical Physics, 2005 **123**: p. 194102

184. T.A. Wesolowski, A. Warshel, J. Phys. Chem., 1993 **97**: 8050.

185. J. Neugebauer, C.R. Jacob, T.A. Wesolowski, E.J. Baerends, J. Phys. Chem. A, 2005 **109**: 7805

186. M.E. Casidas, T.A. Wesolowski, Int. J. Quantum Chem., 2004 **96**: 577.

187. T.A. Wesolowski, J. Am. Chem. Soc. 2004 **126**: 11444.

188. T.A. Wesolowski, J. Chem. Phys. 1997 **106**: 8516.

189. C.R. Jacob, T.A. Wesolowski, L. Visscher, J. Chem. Phys., 2005 **123**: 174104

190. C.R. Jacob, J. Neugebauer, L. Jensen, L. Visscher, Phys. Chem. Chem. Phys., 2006 **8**: 2349

191. J. Neugebauer, M.J. Louwerse, E.J. Baerends, T.A. Wesolowski, J. Chem. Phys., 2005 **122**: 094115

192. J. Neugebauer, M.J. Louwerse, P. Belanzoni, T.A. Wesolowski, E.J. Baerends, J. Chem. Phys., 2005 **123**: 114101

193. J. Neugebauer, E.J. Baerends, J. Phys. Chem. A, 2006, accepted.

194. L.H. Thomas, Proc. Cambridge Philos. Soc., 1927 **23**: 542.

195. E. Fermi, Z. Phys., 1928 **48**: 73.

196. C.F. von Weizsiäcker, Z. Phys., 1935 **96**: 431.

197. A. Lembarki, H. Chermette, Phys. Rev. A, 1994 **50**: 5328.

198. H. Lee, C. Lee, R.G. Parr, Phys. Rev. A, 1991 **44**: 768.

199. J.P. Perdew, Wang Yue, Phys. Rev. B, 1986 **33**: 768.

200. H. Ou-Yang, M. Levy, Int. J. Quantum Chem., 1991 **40**: 379.

201. A.J. Thakkar, Phys. Rev. A, 1992 **46**: 6920.

202. J. Neugebauer, E. J. Baerends, E. Efremov, F. Ariese, C. Gooijer, J. Phys. Chem. A, 2005 **109**: 2100.

203. J. Neugebauer, E. J. Baerends, M. Nooijen, J. Chem. Phys. 2004, **121**: 6155.

204. J. Neugebauer, Vibronic Coupling Calculations using ADF, documentation on the VIBRON module available on request.

205. T.A. Wesolowski, in: Computational Chemistry: Reviews of Current Trends - Vol. 10, World Scientific, 2006 in press.

206. M. Zbiri, M. Atanasov, C. Daul, J.-M. Garcia Lastra, and T.A. Wesolowski, Chem. Phys. Lett. 2004, **397**: 441.

207. M. Zbiri, C.A. Daul., T.A. Wesolowski, J. Chem. Theor. and Comput. 2006, accepted.

208. A. Berces, R. M. Dickson, L. Fan, H. Jacobsen, D. Swerhone, T. Ziegler, Computer Physics Communications, 1997, **100**: 247.

209. H. Jacobsen, A. Berces, D. Swerhone, T. Ziegler, Computer Physics Communications, 1997, **100**: 263.

210. S. K. Wolff, Int. J. Quantum Chem., 2005, **104**: 645.

211. S. Grimme, J. Comp. Chem., 2004, **25**: 1463.

212. M. Ernzerhof, G. Scuseria, J. Chem. Phys., 1999, **110**: 5029.

213. C. Adamo, V. Barone, J. Chem. Phys., 1999, **110**: 6158.

214. A.D. Buckingham, P.W. Fowler, J.M. Hutson, Chem. Rev., 1988, **88**: 963.

215. J.M. Ducere, L. Cavallo, University of Salerno, Italy, article in preparation.

Keywords

A1FIT [1]
ADDDIFFUSEFIT [1]
ALLOW [1]
ALLPOINTS [1]
ANALYTICALFREQ [1]
ATOMS [1] [2] [3]
BASIS [1]
BONDORDER [1]
CDSPECTRUM [1]
CHARGE [1]
CINEB [1]
COLLINEAR [1]
COMMENT [1]
COMMTIMING [1]
CONSTRAINT [1]
COREPOTENTIALS [1]
CREATE [1] [2] [3] [4]
CURRENTRESPONSE [1]
DEBUG [1]
DEFINE [1]
DENSPREP [1]
DEPENDENCY [1]
DIPOLEMAT [1]
DISK [1]
DRF [1]
EFIELD [1]
ENERGYFRAG [1]
EPRINT [1]
EPSFIT [1]
ESR [1]
EXACTDENSITY [1]
EXCITATIONS [1]
EXTENDEDPOPAN [1]
EXTERNALS [1]
FDE [1]
FILE [1]
FITELSTAT [1]
FORCEALDA [1]
FRAGMENTS [1]
FRAGOCCUPATIONS [1]
FREQUENCIES [1]
FULLFOCK [1]
FULLSCF [1]
GEOMETRY [1] [2]
GEOSTEP [1]
GEOVAR [1] [2]
HARTREEFOCK [1]
HESSDIAG [1]
HESSTEST [1]
HFEXCHANGE [1]
HYPERPOL [1]
INLINE [1]
INTEGRATION [1] [2] [3]
IRC [1]
IRCSTART [1]
KEY [1]
LINEARSCALING [1]
LINEARTRANSIT [1]
LOCORB [1]
METAGGA [1]
MMDISPERSION [1]
MODIFYEXCITATION [1]
MODIFYSTARTPOTENTIAL [1]
NONCOLLINEAR [1]
NOPRINT [1]
NOSAVE [1]
OCCUPATIONS [1]
OLDGRADIENTS [1]
OPTICALROTATION [1]
PRINT [1]
QTENS [1]
RADIALCOREGRID [1]
RAMAN [1]
RELATIVISTIC [1]
REMOVEFRAGORBITALS [1]
RESPONSE [1]
RESRAMAN [1] [2]
RESTART [1]
RESTRAINT [1]
SAVE [1]
SCF [1]
SFTDDFT [1]
SICOEP [1]
SINGULARFIT [1]
SKIP [1]
SLATERDETERMINANTS [1]
SMOOTH [1]
SOLVATION [1]
STCONTRIB [1]
STOPAFTER [1]
SYMMETRY [1]
TAILS [1]
TDA [1]
THERMO [1]
TITLE [1]
TRANSITIONSTATE [1]
UNITS [1]
UNRESTRICTED [1]
VANDERWAALS [1]
VECTORLENGTH [1]
VIBRON [1]
XC [1]

Index

A-tensor [1]
adf2aim [1]
force constants [1]
fragment mode [1]
PBE [1]
PBE0 [1]

adfnbo [1]
AIM [1]
alternative elements [1]
analytic second derivatives [1]
atomic coordinates [1]
atomic database [1] [2]
atoms in molecules [1]
automatic mode [1]
B1LYP [1]
B1PW91 [1]
B3LYP [1]
B3LYP* [1]
Bader's analysis [1]
BAS [1]
basic atoms [1]
basis functions [1]
basis set superposition error [1]
basis sets [1] [2]
BHandH [1]
BHandHLYP [1]
BLYP [1]
bond energy analysis [1] [2] [3]
bond order [1] [2] [3] [4]
BP86 [1]
broken symmetry [1]
BSSE [1]
C6 coefficient [1]
Cartesion functions [1]
CD spectrum [1]
charge analysis [1]
CINEB [1]
circular dichroism [1]
climbing-image nudged elastic band [1]
collinear [1]
constrained optimizations [1] [2] [3]
constrained space orbital variation [1]
convergence problems [1]
core excitations [1]
core potential [1]
COSMO [1]
create mode [1]
fragment orbitals [1]
fragments [1]
fragments files [1]
frequencies [1]
frozen core approximation [1]
frozen-density embedding [1]
g-tensor [1] [2]
gennbo [1]
geometry optimization [1]
GGA functionals [1]
ghost atoms [1]
GRAC [1]
Hartree-Fock (post SCF) [1]
Hartree-Fock (SCF) [1] [2]
Hessian [1]
Hirshfeld charges [1] [2]
homogeneous electric field [1]
hybrid functionals (post SCF) [1]
hybrid functionals (SCF) [1] [2]
hyperfine interaction [1]
hyperpolarizability [1] [2]
imaginary frequencies [1]
infrared frequencies [1]
infrared intensities [1]
initial Hessian [1]
internal coordinates [1]
intrinsic reaction coordinate [1]
IR frequencies [1]
IRC [1]
irreducible representation [1]
isotope shift [1]
KMLYP [1]
KT1 [1]
LB94 [1]
LDA functionals [1]
linear dependency [1]
linear scaling techniques [1]
linear transit [1]
localized orbitals [1]
LT (linear transit) [1]
Mayer bond order [1] [2]
Perdew-Zunger SIC [1]
periodic table [1]
point charges [1]
polarizability [1] [2]
population analysis [1]
precision [1]
precision SCF [1]
pseudopotentials [1]
PW91 [1]
Q-tensor [1]
QM/MM [1] [2]
quadrupole moment [1]
Raman (resonance) [1]
Raman intensities [1]
Raman scattering [1]
reaction path [1]
reduction of output [1]
relativistic core potentials [1]
relativistic effects [1]
remove fragment orbitals [1]
resonance Raman [1]
response properties [1]
restart file [1]
restrained optimizations [1]
revPBE [1]
RPBE [1]
run types [1]
SAOP [1]
Schönflies symbol [1]
self-interaction correction [1]
SFO [1]
SFO population analysis [1]
SIC potentials [1]
singlet-singlet excitations [1]
singlet-triplet excitations [1]
smeared occupations [1]
smoothing of gradients [1]
solvent effects [1] [2]
spin [1]
spin-flip excitations [1]
spin-orbit coupling [1]

CSOV analysis [1]
Davidson algorithm [1]
debug [1]
delocalized coordinates [1]
density fitting [1]
dependency [1]
DIIS [1]
dipole allowed [1]
dipole moment [1]
discrete solvent RF model [1]
dispersion (MM) [1]
dispersion coefficients [1]
double group symmetry [1]
doublet-doublet excitations [1]
doublet-quartet excitations [1]
DRF [1]
EFG [1]
electric field (homogeneous) [1]
electric field gradient [1]
electron paramagnetic resonance [1] [2]
electron smearing [1]
electron spin resonance [1] [2]
electronic configuration [1] [2] [3]
end input [1]
EPR [1] [2]
ESR [1] [2]
exchange-correlation [1]
excitation energies [1]
excitation energies spin-orbit [1]
execution of ADF [1]
FDE [1]
fit functions [1]
MDC [1] [2]
memory usage [1]
meta-GGA functionals [1]
minimal input [1]
MM dispersion [1]
model potentials [1] [2]
MOPAC Z-matrix [1]
mPBE [1]
mPW [1]
mPW1K [1]
mPW1PW [1]
Mulliken population [1]
Mulliken poulation [1]
multiplet states [1]
multipole derived charges [1] [2]
Nalewajski-Mrozek bond order [1]
NBO-analysis [1]
NMR chemical shifts [1]
NMR spin-spin coupling [1]
non-collinear [1]
NQCC [1]
numerical integration [1]
O3LYP [1]
OLYP [1]
OPBE0 [1]
open shell TDDFT [1]
optical rotation (dispersion) [1]
orbital localization [1]
ORD [1]
orthonormal basis [1]
parallel version [1]
Pauli Hamiltonian [1]
spin-orbit TDDFT [1]
spin-polarized calculation [1]
STO [1]
STO basis sets [1] [2]
subspecies [1]
symmetry [1]
symmetry label [1]
TAPE13 [1]
TAPE21 [1]
TDCDFT [1]
TDDFT [1]
TDDFT SO [1]
thermodynamics [1]
time-dependent current DFT [1]
time-dependent DFT [1]
transition state [1]
trouble shooting [1]
TS (transition state) [1]
unrestricted calculation [1]
unrestricted fragments [1]
UV/Vis [1]
van der Waals interaction [1] [2]
VDD charges [1] [2]
Voronoi deformation density [1] [2]
VWN [1]
X-ray photoelectron spectroscopy [1]
X3LYP [1]
XC [1]
XPS [1]
Z-matrix coordinates [1]
Zeeman interaction [1] [2]
ZORA [1]